December 8, 2014
version 0.1

GammaCombo User Manual

Till Moritz Karbach[1]

[1] *European Organization for Nuclear Research (CERN), Geneva, Switzerland*

Abstract

GammaCombo is a framework to combine measurements in order to compute confidence intervals on parameters of interest. A global likelihood function is constructed from the probability densitiy functions of the input observables, which is used to derive likelihood-based intervals and frequentist intervals based on pseudoexperiments following the Plugin method to handle nuisance parameters.

Contents

1 Introduction

The interpretation of observables in terms of parameters of the underlying theory is a common problem in physics. Well established methods exist on how to do that, but in practice, their application is often found to be cumbersome. GammaCombo is a framework that hopes to turn this process into a more pleasurable experience.

The basic principle is to assume a probablility density function (PDF) for the observables, and form a likelihood function. From these, best-fit values as well as confidence intervals or upper limits can be computed, following two staticical methods: the profile likelihood construction and the Plugin method. Both methods are located at the more frequentist-like side of the spectrum of statistical schools.

The strength of GammaCombo is to provide tools helping in the following areas:

- provide a framework to construct the likelihood function
- implement minimization strategies to aid the profiling of nuisance parameters
- provide debug tools helping to identify problems during profiling
- make publication quality plots, both one and two dimensional
- provide a way to run large scale toy experiments on a batch system
- provide debug tools helping to judge the quality of frequentist toys
- combine likelihood functions corresponding to different measurements

While being able to save the user a lot of tedious work, one of the design choices is to not invent a black box that is just too dark. After all, applying statistical methods is, in most cases, not trivial and requires a detailed understanding of what is going on. On the downside, this requires the user to code in `C++`.

GammaCombo is able to run state-of-the art sized combinations of measurements. An example of such combinations is LHCb's combination of measurements that are sensitive to the CKM angle γ [1].

2 Statistical methods

2.1 Basic Principle

The strategy to combine the input measurements described in the previous sections is to build a combined likelihood function. This combined likelihood is the product of the individual likelihoods. Denoting the observables by capital Latin letters, and parameters by Greek letters, the combined likelihood $\mathcal{L}$ is

$$\mathcal{L}(\vec{\alpha}) = \prod_i \mathcal{L}_i(\vec{\beta}_i) \ , \tag{1}$$

where the product runs over all input measurements, the vectors $\vec{\beta}_i$ hold the parameters of the input measurements, the vector $\vec{\alpha}$ holds all parameters of the $\vec{\beta}_i$, and the $\mathcal{L}_i$ are the input likelihoods. In this context, the likelihood and the probability density function (PDF) are very similar. The PDF, say f, is a function not only of the parameters $\vec{\beta}$, but also of the observables $\vec{A}$:

$$f = f(\vec{A}|\vec{\beta}) \ . \tag{2}$$

The likelihood is obtained by fixing the observables to the measured values. If multiple measurements of the same kind were performed, the likelihood is the product of as many PDF terms. But in the context of the combination, each measurement was performed exactly once, thus the distinction is subtle,

$$\mathcal{L}(\vec{\beta}) \equiv f(\vec{A}|\vec{\beta})\Big|_{\vec{A}=\vec{A}_{\rm obs}} \ . \tag{3}$$

To give an example, lets consider the combination of two measurements. The first is a cartesian measurement, which measures four cartesian observables that are related to polar coordinates through

$$x_\pm = r_B^{DK} \cos(\delta_B^{DK} \pm \gamma) \tag{4}$$
$$y_\pm = r_B^{DK} \sin(\delta_B^{DK} \pm \gamma) \, . \tag{5}$$

The second measurement is a direct measurement of the parameter r_B^{DK},

$$(r_B^{DK})_{\rm obs} = r_B^{DK} \, . \tag{6}$$

Therefore the vector of parameters $\vec{\alpha}$ has the following components:

$$\vec{\alpha} = (\gamma, r_B^{DK}, \delta_B^{DK})^T \ , \tag{7}$$

where T means transpose. The purpose of the combination is to find the best values of the components of $\vec{\alpha}$, defined as those that maximize the combined likelihood, and to compute well-defined error intervals. The vector of observables, $\vec{A}$, has the following form:

$$\vec{A} = (x_-, y_-, x_+, y_+, (r_B^{DK})_{\rm obs})^T \ . \tag{8}$$

The components of $\vec{A}$ are related to those of $\vec{\alpha}$ by the truth relations Eqns. 4 and 6. We can now construct the combined likelihood as

$$\mathcal{L}(\vec{\alpha}) = f(\vec{A}_{\text{obs}}|\vec{A}(\vec{\alpha})) \ , \tag{9}$$

where $\vec{A}_{\text{obs}}$ holds the measured results, and $\vec{A}(\vec{\alpha})$ are the truth relations. In case of uncorrelated observables, this factors into

$$\mathcal{L}(\vec{\alpha}) = \prod_i f_i(\vec{A}_{i,\text{obs}}|\vec{A}_i(\vec{\alpha_i})) \ . \tag{10}$$

To obtain confidence intervals, a first and simple approach is to inspect the profile likelihood curve, which is described in Section 2.2. This has the advantage of being computationally cheap, but in many examples comes at the price of bad frequentist coverage. A better method is to rely on a method based on pseudo experiments, the PLUGIN method, which is described in Section 2.3. This method is much more computing expensive. It builds upon the profile likelihood method, so in any case it does make sense to start with the easier PROB method.

2.2 The profile likelihood method (PROB)

We define a χ^2-function as

$$\chi^2(\vec{\alpha}) = -2 \ln \mathcal{L}(\vec{\alpha}). \tag{11}$$

In general, there are many equivalent global minima, reflecting the fact that there are multiple solutions possible. At each of these minima, the χ^2 has a value χ^2_{min}. To evaluate the confidence level of a certain truth parameter (for example γ) at a certain value (γ_0) we consider the value of the χ^2-function at the new minimum, $\chi^2_{\text{min}}(\vec{\alpha}')$, where the γ component of $\vec{\alpha}'$ is fixed to $\gamma = \gamma_0$. The new minimum satisfies $\Delta\chi^2 = \chi^2_{\text{min}}(\vec{\alpha}') - \chi^2_{\text{min}} \geq 0$. In a purely Gaussian situation for the truth parameters, the p value $p \equiv 1 - \text{CL}$ is given by the probability that $\Delta\chi^2$ is exceeded for a χ^2-distribution with one degree of freedom:

$$1 - \text{CL} = \frac{1}{\sqrt{2}\,\Gamma(1/2)} \int_{\Delta\chi^2}^{\infty} e^{-t/2} t^{-1/2} \, \mathrm{d}t. \tag{12}$$

Because Eq. (12) can be evaluated using the `TMath::Prob()` function,

$$1 - \text{CL} = \text{Prob}(\Delta\chi^2, n_{\text{dof}} = 1) \ , \tag{13}$$

this method is sometimes called the "PROB method", for example described in the PDG booklet [2]. It is equivalent to the profile likelihood method, implemented e.g. in the MINOS method of MINUIT. The confidence intervals are obtained from the $1 - \text{CL}$ curves by finding the point where it intersects the $1 - \text{CL} = 1 - 0.68$ and $1 - 0.95$ levels. These points are found by interpolating between the four nearest scan points. The PROB method is known to undercover in many situations, that is, the reported intervals contain the true value in a lesser number of times than claimed. However it can also overcover, in which case the reported intervals are too large.

2.3 The pseudo-experiment based frequentist method (PLUGIN)

A more accurate approach is to not rely on the linearity of the truth relations, that relate the observables to the parameters, for example Eqns. 4 and 6. Instead on assuming that the test statistics $\Delta\chi^2$ is distributed as a χ^2, one should compute its distribution using pseudo experiments. This method is based on the approach by Feldman and Cousins [3], and extends it by introducing the concept of nuisance parameters. In the "PLUGIN" method [4] (sometimes also "$\hat{\mu}$ method"), the nuisance parameters are kept, at all times, at their best fit values determined by the observed data, that is they are kept at their profile likelihood point. The algorithm of the method is equivalent to the algorithm propsed by Feldman and Cousins in the absence of nuisance parameters [5]. It is briefly described in the following. For a certain value of interest (γ_0), we:

1. calculate $\Delta\chi^2 = \chi^2_{\text{min}}(\vec{\alpha}') - \chi^2_{\text{min}}$ as before;
2. generate a "toy" result $\vec{A}_{\text{toy}}$, using Eq. 3 with parameters set to $\vec{\alpha}'$ as the PDF;
3. calculate $\Delta\chi^{2\prime}$ of the toy result as in the first step by replacing $\vec{A}_{\text{obs}} \to \vec{A}_{\text{toy}}$, *i.e.* minimize again with respect to $\vec{\alpha}$, once with the scan parameter floating and once with the scan parameter fixed;
4. calculate $1-\text{CL}$ as the fraction of toy results which perform worse than the measured data, *i.e.*
$$1 - \text{CL} = N(\Delta\chi^2 < \Delta\chi^{2\prime})/N_{\text{toy}} \ . \tag{14}$$

It has better coverage properties than the PROB method, but also not necessarily a perfect coverage. The reason is, that at each scan step the nuisance parameters (components of $\vec{\alpha}$ other than the scan parameter) are plugged in at their best fit values for this step, as opposed to computing an n-dimensional Neyman confidence belt. In our simple example we have already $n = 3$, but situations with many more parameters are very common. This would render a fine grained scan of the parameter space computationally very expensive.

In our experience the intervals obtained using the PLUGIN method are usually within 10%–20% of those obtained using the PROB method. This means there are only few cases where qualitative statements obtained using the PROB method become obsolete when considering the PLUGIN results.

3 Installation

The installation of GammaCombo requires the version management system `git` and the build system `cmake` to be installed on your system. It also requires an installation of `Root`, compiled with the `RooFit` library, and a C++ compiler recent enough to support C++11. The source code is hosted on `github.com`. The following steps install GammaCombo on your system.

```
# get and build gammacombo
git clone https://github.com/mkarbach/gammacombo.git
cd gammacombo
mkdir build
cd build
cmake ..
make # can use -j4 to use 4 CPU cores
make install
```

In addition to the GammaCombo core library, this also builds an example combiner called tutorial. To run a first combination, try the following:

```
# run the tutorial
cd tutorial
bin/tutorial -u # usage
bin/tutorial -c 1 --var a_gaus -i # run combination 1
ls -l plots/pdf/tutorial_tutorial_1_a_gaus.pdf # the plot that was created
```

More on the tutorial is described later in Sec. 4. The GammaCombo source code is documented with `doxygen`. You can build the `doxygen` web page documentation, if you have `doxygen` installed, and probably also `graphviz`'s `dot` command. The documentation is created in the `doc` subdirectory of the top level directory.

```
# build the doxygen documentation
# still in the build directory
make doc
# open the created webpages on Mac:
cd ..; open doc/html/index.html
```

4 Tutorial

The tutorial consists of three parts: a simple Gaussian, a two-dimensional Gaussian, and a circular PDF that puts a constraint on a radius parameter. Lets run the tutorial executable to get an overview of what PDFs and combinations are defined in the tutorial (`main/tutorial.cpp`).

```
$ bin/tutorial -u # -u always prints the usage and exits

Available PDFs:
===============
 ( 1) 1D Gaussian (A)
 ( 2) 1D Gaussian (B)
 ( 3) 2D Gaussian (A,B)
 ( 4) circle (A,B)

Available combinations (to be used as -c <n>):
==============================================
 (0) empty
 (1) Gaus 1
 (2) Gaus 2
 (3) Gaus 1 & Gaus 2
 (4) 2D Gaus
 (5) 2D Gaus & Gaus 1
 (6) Circle
 (7) 2D Gaus & Circle
 (8) Gaus 1 & Gaus 2 & 2D Gaus
```

4.1 Simple Gaussian measurement

We start with the probably most simple case of all: a theory parameter being constrained through a single measurement, that follows a simple Gaussian PDF. To find the best estimate for the theory parameter a_{th} the following likelihood function is maximized:

$$\mathcal{L}(a_{\text{th}}) = G(a_{\text{obs}}, \sigma_{a_{\text{obs}}}, a_{\text{th}}) = \frac{1}{\sigma_{a_{\text{obs}}}\sqrt{2\pi}} \exp\left(\frac{-(a_{\text{obs}} - a_{\text{th}})^2}{2\sigma^2_{a_{\text{obs}}}}\right) . \tag{15}$$

Here, the result of the measurement is denoted as

$$a_{\text{obs}} \pm \sigma_{a_{\text{obs}}} , \tag{16}$$

and the numerical values used in the tutorial's PDF 1 are

$$a_{\text{obs}} = -0.5 \pm 1.0 . \tag{17}$$

The resulting confidence intervals for a_{th} are computed with

```
cd tutorial
bin/tutorial -c 1 -i --var a_gaus --ps 1
```

where the options mean

`-c` combination ID. This refers to pre-defined combinations (in this case nothing is combined, really, it is just the simple Gaussian). A list of available IDs can be obtained by running with the usage flag, `bin/tutorial -u`.

`-i` interactive mode. Any plot will be shown directly in the familiar ROOT canvas. Exit with `Ctrl+c`.

`--var` the variable name of the theory parameter to be scanned.

`--ps 1` print solution on the plot. The given value configures the position of the numerical value on the plot. See the help `bin/tutorial -h` for options.

The resulting plot is shown in Fig. 1.

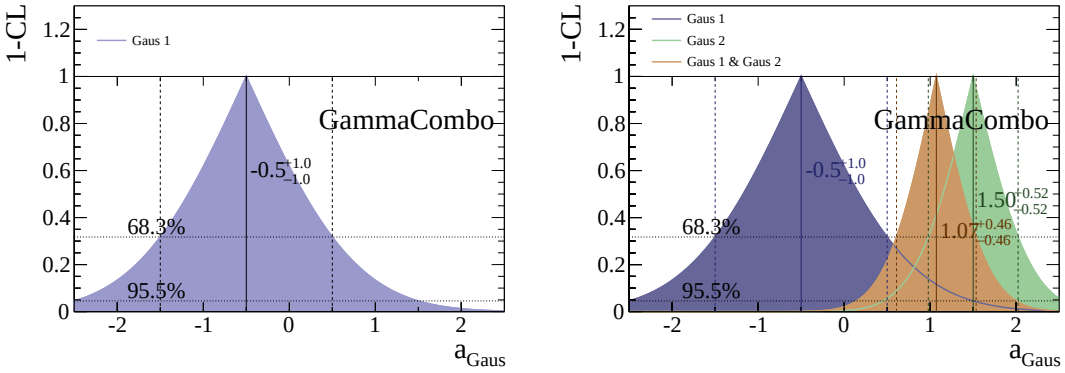

Figure 1: Left: Result of the tutorial 1, a simple Gaussian.

4.2 Combining two Gaussian measurmeents

In the tutorial executable, there is also a second Gaussian defined, PDF 2, which constrains a_{obs} to a different value,

$$a_{\text{obs}} = 1.0 \pm 0.5\,(\text{stat}) \pm 0.15\,(\text{syst})\,. \tag{18}$$

The combination of Eq. 17 and 18 should reproduce the usual weighted average,

$$a_{\text{th}} = \frac{w_1 a_1 + w_2 a_2}{w_1 + w_2} \tag{19}$$

$$\sigma_{\text{th}} = \frac{1}{\sqrt{w_1 + w_2}} \tag{20}$$

$$w_i = \frac{1}{\sigma_{a_i}^2}\,, \tag{21}$$

where the a_i refer to Eq. 17 and 18, and the statistical and sytematic uncertainty of Eq. 18 was combined in quadrature. The numerical result of Eq. 19 is

$$a_{\text{th}} = 1.07 \pm 0.46\,. \tag{22}$$

Lets put them both into the same plot, and then also plot the combination of both. The resulting plot is shown in Fig. 1.

```
# multiple instances of -c will be added to the same plot
bin/tutorial -c 1 -c 2 -i --var a_gaus
# combination number 3 combines both Gaussians
bin/tutorial -c 1 -c 2 -c 3 -i --var a_gaus --ps 1
```

Options used:

`-c 1 -c 2 -c 3` When the combination argument is given multiple times, all combinations will be computed and added to the same plot. The order controls the order in the plot, where the last one is in the foreground.

4.3 Two-dimensional Gaussian measurement

The next PDF of the tutorial, PDF 3, is a two-dimensional Gaussian in the previous variable, a, and another one, b. Now the likelihood function is given by

$$\begin{aligned}\mathcal{L}(a_{\text{th}}, b_{\text{th}}) &= G(a_{\text{obs}}, \sigma_{a_{\text{obs}}}, a_{\text{th}}, b_{\text{obs}}, \sigma_{b_{\text{obs}}}, b_{\text{th}}) &(23)\\ &= \frac{1}{\sqrt{\det V}(\sqrt{2\pi})^{n/2}} \exp\left(-\frac{1}{2}(\vec{x}_{\text{obs}} - \vec{x}_{\text{th}})^T V^{-1} (\vec{x}_{\text{obs}} - \vec{x}_{\text{th}})\right)\,, &(24)\end{aligned}$$

where $\vec{x} = (a_{\text{obs}}, b_{\text{obs}})$ is the vector of observables, n is the number of observables, and V is the covariance matrix. Combination number 4 contains only this two-dimensional measurement, the numerical values of the observations are

$$\begin{aligned}a_{\text{obs}} &= 0.1 \pm 1.0\,, &(25)\\ b_{\text{obs}} &= 1.5 \pm 1.0\,, &(26)\\ \rho(a, b) &= 0.6\,. &(27)\end{aligned}$$

The resulting plots are shown in Fig. 2.

```
# the 2D scan is triggered by giving a second --var argument
bin/tutorial -c 4 --var a_gaus --var b_gaus -i
# redo the scan to print 2D confidence contours and a
# marker at the best fit value
bin/tutorial -c 4 --var a_gaus --var b_gaus -i --npoints2d 70 --2dcl --ps 1
# just remake the plot (without redoing the scan)
# activating magnetic plot boundaries
bin/tutorial -c 4 --var a_gaus --var b_gaus -i --2dcl --ps 1 --magnetic -a
    plot
```

The command line options mean:

`--var a_gaus --var b_gauss` specify the two variables to perform the scan for. The variable given first will be plotted on the horizontal axis, the second one on the vertical axis.

`--2dcl` plot contours corresponding to "two-dimensional" confidence level. The default is to plot contours such that their projection to the axes coincides with the one-dimensional scans. In this case the resulting innermost contour corresponds to 39% CL. When the option is given, the innermost contour is enlarged to hold 68% CL.

`--npoints2d 70` use 70 scan points in each direction, so $70^2 = 4900$ in total.

`--ps 1` print a marker at all local minima. If `--ps 2` is given, only the best fit value is indicated.

`--magnetic` switches on magnetic plot boundaries, that will drag the contours towards them. In many cases, this results in nicer contours, but not always.

`-a plot` activates the plot action: Whenever a scan is done, the result is saved into a file in `plots/scanner`. This can be read in to remake the plot, without having to rerun the entire scan. With more complex combinations, this saves time.

While a two-dimensional scan is running in interactive mode (`-i`), a window opens illustrating the scan progress. The scan is done in a trajectory spiralling away from the start points. In this plot, the current $\Delta\chi^2$ is plotted. Any subsequent scan opens one more of these plots. If in a subsequent scan lower $\Delta\chi^2$ values are encountered, this will show up. At the end of the scan, one more window is opened showing the final $\Delta\chi^2$, from which the confidence contours are computed. Updating these control plots can consume a significant computing time for simple combinations, like the tutorial. In these cases, the program runs much faster non-interactively (without `-i`).

For more complex combinations, two-dimensional scans can take a long time, especially with a high granularity. For these cases, the `-a plot` option is very useful, which allows the plot to be redone without having to rerun the scan. This way, the look and feel can be changed: plot markers at best fit points, change titles, change colors, change the number of contours, activate magnetic boundaries, etc. For this, the only important options are `-c 6 --var a_gaus --var b_gaus`, all other ones are not needed to identify the correct file and can be omitted.

4.4 A more advanced example: circular constraint

The next measurement of the tutorial is a circular constraint for the parameters a and b. This happens if we have a measurement of a radius-like parameter. In PDF 4 of the tutorial, a measurement of r is implemented,

$$r = 2.0 \pm 0.25\,, \tag{28}$$

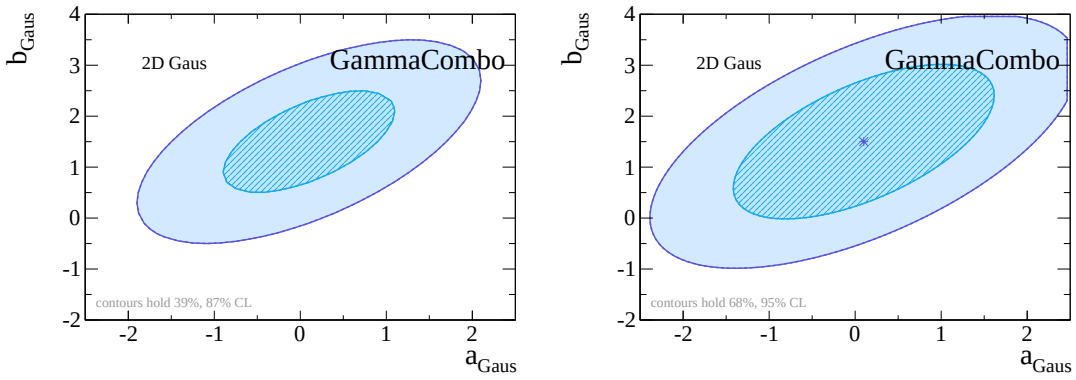

Figure 2: Result of the tutorial 4, a two-dimensional Gaussian. Left: with "1D" confidence regions. Right: with "2D" confidence regions and a marker at the best fit value.

which is related to the parameters a and b through the truth relation

$$r = \sqrt{a^2 + b^2}\,. \tag{29}$$

The constraint is shown in Fig. 3 (left). Let's combine this with the two-dimensional Gaussian measurement from the previous example. At first, both could be added to the same plot Fig. 3 (right). Then, we could combine the circular constraint with the two-dimensional Gaussian measurement and plot all three combinations together. This is shown in Fig. 4.

```
# scan the circular constraint of the tutorial
bin/tutorial -c 6 --var a_gaus --var b_gaus -i
# add the previous 2D Gaussian to the same plot
bin/tutorial -c 6 --var a_gaus --var b_gaus -a plot -i -c 4
# combine the circular constraint with the two-dimensional Gaussian
bin/tutorial -c 7 --var a_gaus --var b_gaus
# plot all three into the same plot
bin/tutorial -c 6 --var a_gaus --var b_gaus -a plot -i -c 4 -c 7
```

4.5 Changing combinations on the fly

It is possible to change existing combinations on the fly from the command line by adding and/or removing PDFs. The PDFs are refered to by their internal numbers, which can be obtained from the usage printout (`-c`).

Example 1: add a PDF Lets add a PDF on the fly to one of the tutorial combinations. The resulting plot is shown in Fig. 5.

```
# This is the unmodified tutorial. Combination 1 consists just of PDF 1.
$ bin/tutorial -i --var a_gaus -c 1
```

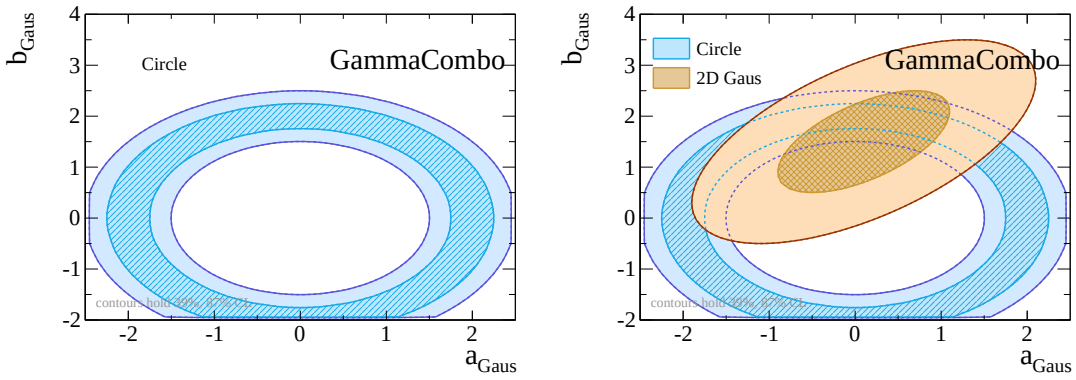

Figure 3: Result of the tutorial 6, a circular constraint. Left: the circular constraint alone. Right: added the two-dimensional Gaussian from the previous example.

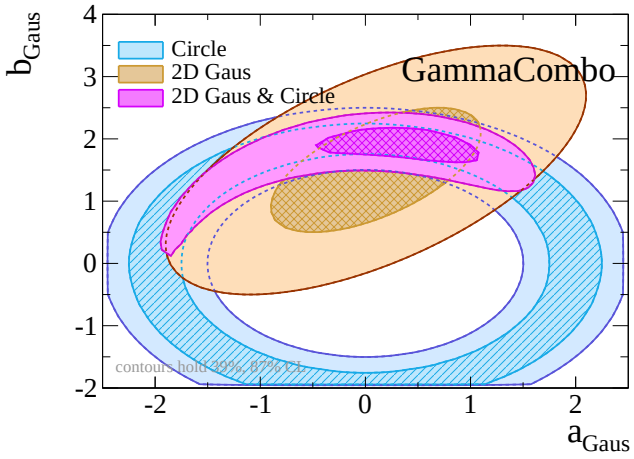

Figure 4: Result of the tutorial 6, a circular constraint. Combining the circular constraint with the two-dimensional Gaussian measurement.

```
# Print out usage to find a PDF number we can add.
$ bin/tutorial -u

Available PDFs:
===============
 ( 1) 1D Gaussian (a_{obs} = -0.5)
 ( 2) 1D Gaussian (a_{obs} = 1.5)
 ( 3) 2D Gaussian (a_{obs}, b_{obs})
 ( 4) circle (a_{obs}, b_{obs})

# The '+2' adds PDF number 2 to the combination.
$ bin/tutorial -i --var a_gaus -c 1:+2
```

```
# Lets add the unmodified combination to the same plot.
$ bin/tutorial -i --var a_gaus -c 1:+2 -c 1
```

Example 2: delete a PDF Lets delete a PDF on the fly from one of the tutorial combinations. The resulting plot is shown in Fig. 5.

```
# This is the unmodified tutorial combination 8 consisting of three PDFs.
$ bin/tutorial -i --var a_gaus -c 8

Combiner Configuration: Gaus 1 & Gaus 2 & 2D Gaus
=======================

 1. [PDF  1] 1D Gaussian (A)
 2. [PDF  2] 1D Gaussian (B)
 3. [PDF  3] 2D Gaussian (A,B)

# Lets delete PDF 1 from the combination.
$ bin/tutorial -i --var a_gaus -c 8:-1 -c 8
```

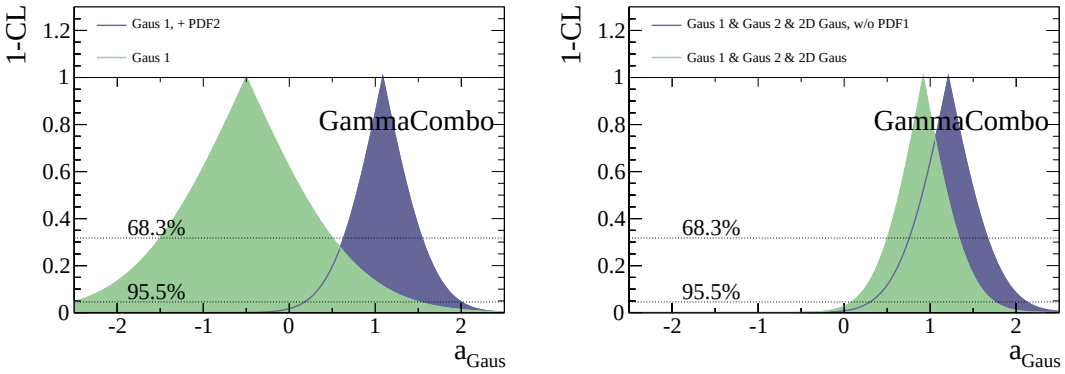

Figure 5: Result of Examples 1 and 2. Left: add a PDF. Right: delete a PDF.

Example 3: a new combiner from scratch An easy trick to allow completely new combinations from scratch is to define an empty combination in the main file. In the tutorial, this combination has index 0. Then one can add whatever one wants.

```
# Adding several PDFs to the empty combination
# to reproduce combination 8:
$ bin/tutorial -i --var a_gaus -c 0:+1,+2,+3
```

4.6 Running the PLUGIN method

In order to run the pseudo-experiment based PLUGIN method for a particular combination requires that a profile-likelihood scan is already set up. Currently, the PLUGIN method

is only implemented for one-dimensional scans. Lets taking as an example combination number 7 ("2D Gaus & Circle") and scan for parameter a. The result is shown in Fig. 6.

```
# Run two jobs to produce toys, possibly on different cores:
$ bin/tutorial -c 7 --var a_gaus -a pluginbatch --ntoys 100 --nrun 1
$ bin/tutorial -c 7 --var a_gaus -a pluginbatch --ntoys 200 --nrun 2
$ bin/tutorial -c 7 --var a_gaus -a pluginbatch --ntoys 400 --nrun 3
# Compute the plugin intervals from all toys:
$ bin/tutorial -c 7 --var a_gaus -a plugin -j 1-3 -i
# Only plot the plugin curve:
$ bin/tutorial -c 7 --var a_gaus -a plugin -j 1-3 -i --po
```

The arguments mean:

`-a pluginbatch` batch mode. This will run a number of PLUGIN toys and save them into an output file. Many of these jobs can be run in parallel for example on a batch farm.

`--ntoys 100` produce 100 PLUGIN toys per scan point.

`--nrun 1` run job number 1. This number will be added to the filenames of the produced PLUGIN toy files. Files will be overwritten if the same number is given.

`-j 1-3` read in PLUGIN toy files number 1 through 3.

`--po` make a PLUGIN-only plot.

The results in Fig. 6 show that, first of all, the PLUGIN method agrees quite nicely with the profile likelihood method. Probably it will yield slightly larger intervals, but in order to be conclusive, one would have to run more toys. Also, the result is visibly non-Gaussian, as it develops an asymmetric tail on the left. This was already apparent from the two-dimensional Fig. 4. In this case, however, the non-Gaussianity likely doesn't affect the statistical coverage, as both methods return quite similar results.

4.7 The standard Feldman-Cousins example

The presence of a physically forbidden region for a parameter complicates the statistical treatment. In particular, the profile likelihood construction is easily spoiled, when the boundary is close to the region of interest. In this situation, the method by Feldman-Cousins provides an accepted frequentist alternative. In the absence of nuisance parameters, the PLUGIN method is equivalent to the method by Feldman-Cousins.

Lets take the example of the simple Gaussian measurement given in Sec. 4.1. The parameter a has a physical limit configured (see `src/ParametersTutorial.cpp`),

$$a > -1.5\,, \tag{30}$$

which has not been enforced so far. It can be activated through the physical range option `--pr`. The resulting plot is shown in Fig. 7 (left). Note that we are trying to scan in the

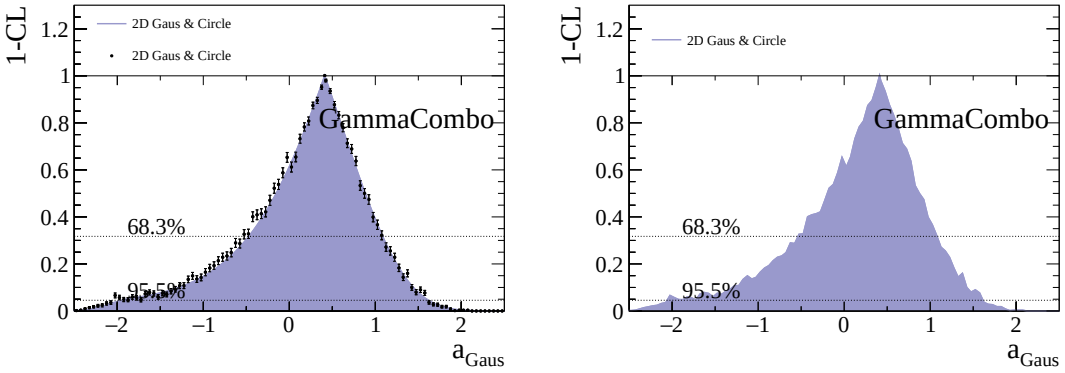

Figure 6: Result of running the PLUGIN method with combination number 7: a 2D Gaussian with a circular constraint.

non-physical region, which is currently not well supported: The drop at $a = -1.5$ is not vertical due to the finite binning, and as a result, the interpolated confidence interval is not accurate. This can be overcome by restricting the scan range to the physical region (Fig. 7 right).

```
# activate physical parameter limits
$ bin/tutorial -c 1 -i --var a_gaus --ps 1 --pr
# restrict scan range to the physically allowed region
$ bin/tutorial -c 1 -i --var a_gaus --ps 1 --pr --scanrange -1.5:2.5
```

The arguments mean:

`--pr` physical range. This will enforce the physical parameter ranges defined in `src/ParametersTutorial.cpp`.

`--scanrange -1.5:2.5` Adjust the scan range to the given range. The default range is defined in `src/ParametersTutorial.cpp`.

Lets now run the PLUGIN toys for this example. The result is shown in Fig. 8. The difference to the profile likelihood method is most apparent close to the physical boundary: the boundary seems to "push away" the lower boundary of the confidence interval. The observed interval boundaries reproduce the nominal ones taken from the original Feldman-Cousins paper [3]. The values to compare to are given in Tab. X of Ref. [3] at $x_0 = 1.0$ because our Gaussian is shifted with respect to theirs by 1.5 units.

```
# run plugin toys for combination 1 (simple Gaussian), with physical boundary
# (try to run mulitple jobs to generate more toys)
$ bin/tutorial -c 1 --var a_gaus -a pluginbatch --ntoys 400 --pr --scanrange
    -1.5:2.5 --nrun 1
# compute the plugin intervals from all toys:
$ bin/tutorial -c 1 --var a_gaus -a plugin -j 1 --pr --scanrange -1.5:2.5 -i
```

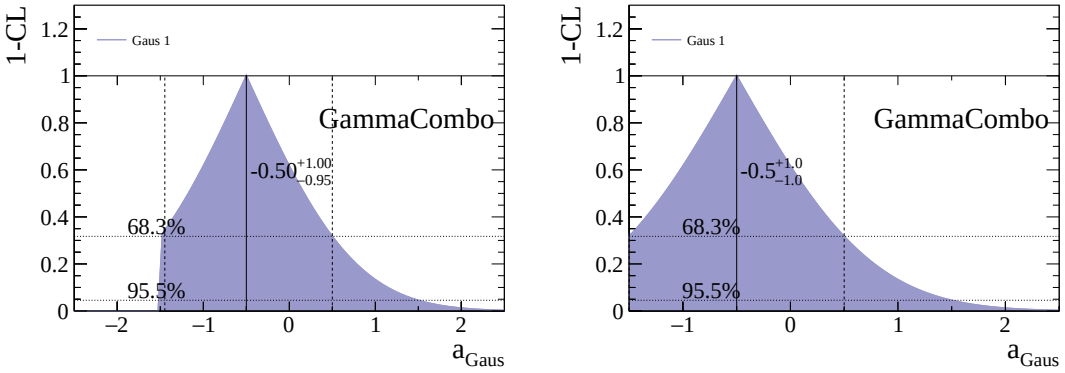

Figure 7: Applying a physical range to combination 1. Left: We tried to scan in the now unphysical region, which caues some inaccurracies (see text). Right: We restricted the scan range to the physical region, resulting in accurately deterimined (but statistically questionable) intervals.

```
a_gaus = [ -1.24,   0.51] ( -0.50 -  0.74 +  1.01) @0.68CL, Plugin
a_gaus = [  -1.5,    1.4] (  -0.5 -   1.0 +   1.9) @0.95CL, Plugin
```

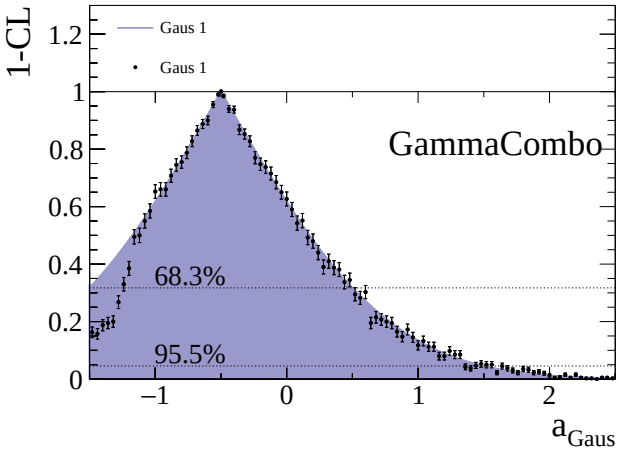

Figure 8: Applying a physical range to combination 1 and running the PLUGIN method. In this case the PLUGIN method is equivalent to the method by Feldman-Cousins. The difference to the profile likelihood method is most apparent close to the physical boundary at $a = -1.5$.

5 Create a new combiner module

In this section we describe how to add a new combiner module to GammaCombo, that will contain everything that is needed for a new combination project: all PDF including the observed data (central values, uncertainties, correlations), parameter definitions, and a set of predefined combinations.

We will take a real-life example based on a measurement of the CKM angle γ, as reported in Ref. [6]. The observables are cartesian coordinates $x_\pm$ and $y_\pm$, which are linked to the parameter of interest, γ, and two nuisance parameters δ_B^{DK} and r_B^{DK} through the relations

$$x_\pm = r_B^{DK} \cos(\delta_B^{DK} \pm \gamma) \tag{31}$$
$$y_\pm = r_B^{DK} \sin(\delta_B^{DK} \pm \gamma)\,. \tag{32}$$

The observables are measured to be

$$x_- = (\ \ 2.5 \pm 2.5 \pm 1.1) \times 10^{-2}\,, \tag{33}$$
$$y_- = (\ \ 7.5 \pm 2.9 \pm 1.5) \times 10^{-2}\,, \tag{34}$$
$$x_+ = (-7.7 \pm 2.4 \pm 1.1) \times 10^{-2}\,, \tag{35}$$
$$y_+ = (-2.2 \pm 2.5 \pm 1.1) \times 10^{-2}\,, \tag{36}$$

where the first uncertainty is statistical, and the second is systematic. The statistical and systematic correlations are given in Tables 1 and 2.

Table 1: Statistical cartesian correlations.

	x_-	y_-	x_+	y_+
x_-	1	-0.247	0.038	-0.003
y_-		1	-0.011	0.012
x_+			1	0.002
y_+				1

Table 2: Systematic cartesian correlations.

	x_-	y_-	x_+	y_+
x_-	1	0.005	-0.025	0.070
y_-		1	0.009	-0.141
x_+			1	0.008
y_+				1

In addition to this real-life cartesian example we will add a mock-up measurement of the nuisance parameter r_B^{DK}, which we can then combine with the cartesian example:

$$r_B^{DK} = 0.10 \pm 0.01\,. \tag{37}$$

1. The following steps describe how to create the source files needed for a new combiner. The complete solution is contained in the directory `tutorial/cartesian`, so the quick way to get it run is to just copy this directory one level up to where the other combiner directories live.

```
cp cartesian/cartesian .
```

   For a more step-by-step like experience, we will copy the existing tutorial directory and modify the source files to make the new cartesian combiner from scratch.

```
cp -r tutorial cartesian
rm -r tutorial/cartesian # delete the solution!
```

2. Tell the GammaCombo build system about the new combiner:

```
vi cmake/combiners.cmake
# edit the file to look like
SET( COMBINER_MODULES
   tutorial
   cartesian
)
vi cartesian/CMakeLists.txt
# change the name of the project to look like this
SET(COMBINER_NAME
   cartesian
)
```

3. Define the project parameters. These will be γ, r_B^{DK}, and δ_B^{DK}. All the parameters are defined in one single class. Here their names are defined, their titles, unit, the default start value for the fit, the default scan range, and the physically allowed region.

```
# rename the parameter class header file to match the new project
cd cartesian/include
mv ParametersTutorial.h ParametersCartesian.h
vi ParametersCartesian.h # edit to change Tutorial for Cartesian
# also rename the implementation file
cd cartesian/src
mv ParametersTutorial.cpp ParametersCartesian.cpp
vi ParametersCartesian.cpp
```

   Add the parameters to the `ParametersCartesian.cpp` file:

```
void ParametersCartesian::defineParameters()
{
   Parameter *p = 0;

   p = newParameter("g");
   p->title = "#gamma";
```

```
    p->startvalue = DegToRad(70);
    p->unit = "Rad";
    p->scan = range(DegToRad(0), DegToRad(180));
    p->phys = range(-7, 7);

    p = newParameter("d_dk");
    p->title = "#delta_{B}^{DK}";
    p->startvalue = DegToRad(127);
    p->unit = "Rad";
    p->scan = range(DegToRad(0), DegToRad(180));
    p->phys = range(-7, 7);

    p = newParameter("r_dk");
    p->title = "r_{B}^{DK}";
    p->startvalue = 0.09;
    p->unit = "";
    p->scan = range(0.02, 0.2);
    p->phys = range(0, 1e4);
}
```

The name is being used on the command line, while the title of a parameter will be used in the plots. The unit of a parameter can be an arbitrary string, however, setting it to `Rad` will tell GammaCombo that this parameter is an angular parameter. During minimization, angular parameters will be kept within the range $[0, 2\pi]$. The start value is being used in the first minimization of the χ^2 function. Some care needs to be taken here, because it is very easy to pick starting values so far away from the sensible region that the χ^2 function reaches too large values that prevent the fit from converging. The default scan range applies to both one- and two-dimensional scans, if not being overwritten from the command line using the options `--scanrange` (horizontal axis) and `--scanrangey` (vertical axis). The physically allowed range can be used to impose hard limits on the value of a parameter, which get activated through the `--pr` option.

4. Create the PDF class holding the measurement of the cartesian observables, as well as the definition of their Gaussian PDF. At first, create the the header file in `include/PDF_Cartesian.h`.

```
#ifndef PDF_Cartesian_h
#define PDF_Cartesian_h

#include "PDF_Abs.h"
#include "ParametersCartesian.h"

using namespace RooFit;
using namespace std;
using namespace Utils;

class PDF_Cartesian : public PDF_Abs
{
```

```
   public:
      PDF_Cartesian(TString cObs="lumi3fb", TString cErr="lumi3fb", TString
          cCor="lumi3fb");
      ~PDF_Cartesian();
      void         buildPdf();
      void         initObservables();
      virtual void initParameters();
      virtual void initRelations();
      void         setCorrelations(TString c);
      void         setObservables(TString c);
      void         setUncertainties(TString c);
};

#endif
```

5. Constructor implementation.

```
   PDF_Cartesian::PDF_Cartesian(TString cObs, TString cErr, TString cCor)
: PDF_Abs(4) // <-- configure the number of observables
{
   name = "cartesian"; // <-- configure the PDF name, should be unique
   initParameters();
   initRelations();
   initObservables();
   setObservables(cObs);
   setUncertainties(cErr);
   setCorrelations(cCor);
   buildCov();
   buildPdf();
}

PDF_Cartesian::~PDF_Cartesian(){}
```

6. Implement the parameter initialization and the truth relations. The truth relations are implemented through `RooForlumaVar` objects. It is also possible to implement them through customized objects, that inherit from `RooAbsReal`. This way, relations of arbitrary complexity can be added. This will be discussed in Sec. 5.1. The names of the predicted observables need to end on `_th`.

```
void PDF_Cartesian::initParameters()
{
   ParametersCartesian p; // <-- use the project's parameter class
   parameters = new RooArgList("parameters");
   parameters->add(*(p.get("r_dk")));
   parameters->add(*(p.get("d_dk")));
   parameters->add(*(p.get("g")));
}

void PDF_Cartesian::initRelations()
{
```

```
   theory = new RooArgList("theory");
   RooArgSet *p = (RooArgSet*)parameters;
   // The order of this list must match that of the COR matrix!
   theory->add(*(new RooFormulaVar("xm_dk_th", "xm_dk_th", "r_dk*cos(d_dk-g)",
       *p)));
   theory->add(*(new RooFormulaVar("ym_dk_th", "ym_dk_th", "r_dk*sin(d_dk-g)",
       *p)));
   theory->add(*(new RooFormulaVar("xp_dk_th", "xp_dk_th", "r_dk*cos(d_dk+g)",
       *p)));
   theory->add(*(new RooFormulaVar("yp_dk_th", "yp_dk_th", "r_dk*sin(d_dk+g)",
       *p)));
}
```

7. Implement the definition of the observables. The names of the observables need to match those of the predicted ones, with the only difference that they end on `_obs`. Their titles will show up in pull plots. The configured value is not important, but the ranges should be far away from any boundary there might be, as they will affect the toy generation within the PLUGIN method.

```
void PDF_Cartesian::initObservables()
{
   observables = new RooArgList("observables");
   // The order of this list must match that of the COR matrix!
   observables->add(*(new RooRealVar("xm_dk_obs", "x-", 0, -1, 1)));
   observables->add(*(new RooRealVar("ym_dk_obs", "y-", 0, -1, 1)));
   observables->add(*(new RooRealVar("xp_dk_obs", "x+", 0, -1, 1)));
   observables->add(*(new RooRealVar("yp_dk_obs", "y+", 0, -1, 1)));
}
```

8. Implement the measured values of the observables. Based on a config string, different values can be loaded. This is very useful if different versions of a given measurement exist, for example a first measurement from the year 2010, and an update from 2012. These config strings are being used inside the main program file, `main/cartesian.cpp` (see below). The string stored in `obsValSource` is meant for information only, it will be printed in the verbose output (`-v`). The "truth" and "toy" entries should not be touched.

```
void PDF_Cartesian::setObservables(TString c)
{
   if ( c.EqualTo("truth") ){
      setObservablesTruth();
   }
   else if ( c.EqualTo("toy") ){
      setObservablesToy();
   }
   else if ( c.EqualTo("lumi3fb") ){
      obsValSource = "arxiv:1408.2748";
      setObservable("xm_dk_obs", 2.5e-2);
      setObservable("ym_dk_obs", 7.5e-2);
```

```
      setObservable("xp_dk_obs",-7.7e-2);
      setObservable("yp_dk_obs",-2.2e-2);
   }
   else {
      cout << "PDF_Cartesian::setObservables() : ERROR : config not found: "
          << c << endl;
      exit(1);
   }
}
```

9. Implement the measured uncertainties of the observables. It is important to match the order that of the observables. The systematic error can be set to zero without problems, but the statistical error should always be finite. The `obsErrSource` string is again solely for informational purposes.

```
void PDF_Cartesian::setUncertainties(TString c)
{
   if ( c.EqualTo("lumi3fb") ){
      obsErrSource = "arxiv:1408.2748";
      StatErr[0] = 0.025; // xm
      StatErr[1] = 0.029; // ym
      StatErr[2] = 0.024; // xp
      StatErr[3] = 0.025; // yp
      SystErr[0] = 0.011; // xm
      SystErr[1] = 0.015; // ym
      SystErr[2] = 0.011; // xp
      SystErr[3] = 0.011; // yp
   }
   else {
      cout << "PDF_Cartesian::setUncertainties() : ERROR : config not found: "
          << c << endl;
      exit(1);
   }
}
```

10. Implement the measured correlations of the observables. Here the order needs to match that of the observables. The function `resetCorrelations()` sets all correlations to zero.

```
void PDF_Cartesian::setCorrelations(TString c)
{
   resetCorrelations();
   if ( c.EqualTo("lumi3fb") ){
      corSource = "arxiv:1408.2748";
      double dataStat[] = {
         // xm     ym     xp     yp
          1.   , -0.247, 0.038, -0.003, // xm
         -0.247, 1.   , -0.011, 0.012, // ym
          0.038, -0.011, 1.  ,  0.002, // xp
         -0.003, 0.012, 0.002, 1.    // yp
```

```
      };
      corStatMatrix = TMatrixDSym(nObs,dataStat);
      double dataSyst[] = {
         // xm      ym      xp      yp
          1.   ,  0.005, -0.025, 0.070, // xm
          0.005,  1.   ,  0.009, -0.141, // ym
         -0.025,  0.009,  1.   ,  0.008, // xp
          0.070, -0.141,  0.008,  1.    // yp
      };
      corSystMatrix = TMatrixDSym(nObs,dataSyst);
   }
   else {
      cout << "PDF_Cartesian::initCorrelations() : ERROR : config not found: "
          << c << endl;
      exit(1);
   }
}
```

11. Implement the Gaussian PDF. It is possible to use any other functional form for the PDF, as will be described in Sec. 5.2, but the Gaussian is probably the most common choice. Don't change the name of this PDF object.

```
void PDF_Cartesian::buildPdf()
{
   pdf = new RooMultiVarGaussian("pdf_"+name, "pdf_"+name,
       *(RooArgSet*)observables, *(RooArgSet*)theory, covMatrix);
}
```

12. Implement the second PDF class for the measurement of r_B^{DK}. We will just adapt the existing class for a simple-Gaussian measurement from the tutorial, `PDF_Gaus`.

    - copy the header file `include/PDF_Gaus.h` into `include/PDF_rb.h`
    - edit the new header file to adjust the class name (`PDF_rb`), and to include the header file of the parameter class of the cartesian project (`ParametersCartesian.h`).
    - copy the implementation file `include/PDF_Gaus.cpp` into `include/PDF_rb.cpp`
    - edit the class name, change to the parameter class of the cartesian project
    - include the paramter `r_dk` rather than `a_gaus`
    - change the name of the r_B^{DK} observable to e.g. `r_dk_obs`; similar for the theory value
    - set the observed central value to 0.1 and the statistical error to 0.01

13. If you created the folder for the cartesian project as suggested by copying the tutorial folder, you should remove now the non-needed classes `PDF_Circle.h`, `PDF_Gaus.h`, `PDF_Gaus2d.h`, to not confuse the build system. Also remove the corresponding implementation files from the `src` directory.

14. Now we need to create the new main file. Rename the one from the tutorial to `main/cartesian.cpp` and change it to look like this:

```
#include <stdlib.h>
#include "GammaComboEngine.h"
#include "PDF_rb.h"
#include "PDF_Cartesian.h"

using namespace std;
using namespace RooFit;
using namespace Utils;

int main(int argc, char* argv[])
{
   GammaComboEngine gc("cartesian", argc, argv);

   // define PDFs
   gc.addPdf(1, new PDF_Cartesian("year2013","year2013","year2013"),
       "Cartesian");
   gc.addPdf(2, new PDF_rb("year2014","year2014","year2014"), "rb");

   // Define combinations
   gc.newCombiner(0, "empty", "empty");
   gc.newCombiner(1, "cartesian1", "Cartesian",  1);
   gc.newCombiner(2, "cartesian2", "rb",         2);
   gc.newCombiner(3, "cartesian3", "Cartesian & rb", 1,2);

   // Run
   gc.run();
}
```

15. Change to the build directory and build the new project

```
cd build
make install -j4
```

16. Run the new combiner. The results are shown in Fig. 9.

```
cd cartesian
bin/cartesian -u
bin/cartesian -c 1 -c 3 --var g -i
bin/cartesian -c 1 -c 2 -c 3 --var r_dk -i
```

5.1 Advanced: C++ truth relations

To be filled.

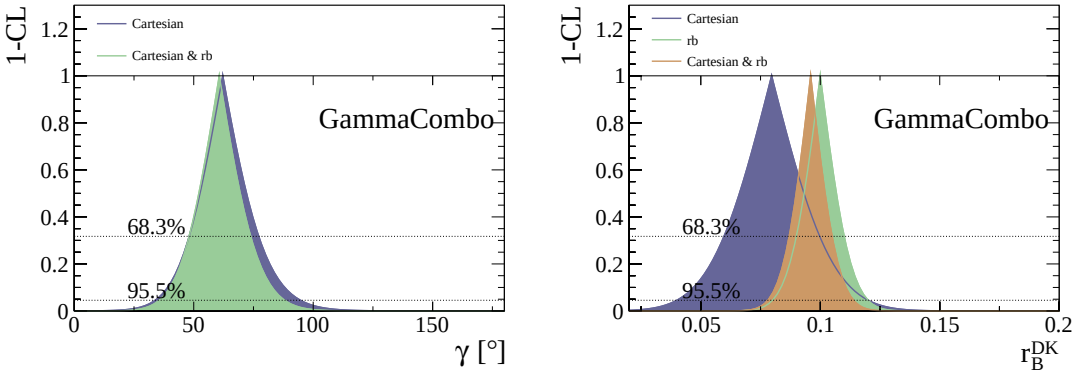

Figure 9: Results of the new combiner module "cartesian". Left: curves for parameter γ. Right: curves for (nuisance) parameter r_B^{DK}.

5.2 Advanced: Non-Gaussian observables

To be filled.

6 Advanced Topics

Most of this section is yet to be filled.

6.1 Defining a custom scan strategy and starting points

6.2 Printing an overview of a combination's structure

6.3 Control Plots

- Giving the option `--pulls` will produce a pull plot showing which observables deviate the most from the prediction. It will also show the value of the χ^2 function at the best fit value, and the corresponding (naive) fit probability.
- Giving the option `-e` will produce a plot of the parameter evolution of a Prob scan.
- GammaCombo produces a graphical representation of the content of a combination in the `.dot` format, that can be interpreted by the `dot` command of the `graphviz` package.

6.4 Run an Asimov toy

6.5 Predict observables

- Observable names are unique in the Combiner so multiple PDFs with observables of the same name can be combined. For this purpose each observable has a unique ID appended, for example "_UID2" for ID 2. The number corresponds to the number that the PDF which provides this observable has in the Combiner.
- One can make predictions for observables by simply scanning for them. When doing this, the corresponding χ^2 constraint usually needs to be tightened. This effectively replaces the inversion of the truth equation by the minimization. A plot is being produced showing the deviation of the actual value of the observable and the predicted one, as a function of the scan step. This should be a flat line at zero, and allows to judge if the constrained was tightened enough.

6.6 Attempt a coverage correction

7 Command line options

Here is a list of available command line options. The full list can always be obtained by running

```
bin/tutorial -h
```

`--2dcl` plot contours corresponding to "two-dimensional" confidence level. The default is to plot contours such that their projection to the axes coincides with the one-dimensional scans. In this case the resulting innermost contour corresponds to 39% CL. When the option is given, the innermost contour is enlarged to hold 68% CL. Sect. 4.3.

`--asimov` run an Asimov toy. Sect. 6.4.

`--color` changes the plot color of the plotted combination. There are six different colors for one-dimensional plots defined in `GammaComboEngine::defineColors()`, that can be specified through numbers 0–5. For two-dimensional plots, there are four different colors defined in `OneMinusClPlot2d::OneMinusClPlot2d()`, that can be specified through the numbers 0–3.

`--controlplots` Make control plot for the PLUGIN toys. Sect. 7.

`--covCorrectPoint` Define the point for the coverage correction (experimental). Sect. 6.6.

`--covCorrect` Activate a coverage correction (experimental). Sect. 6.6.

`--digits, -s` Set the number of printed digits right of the decimal point. Default is automatic.

`--evol, -e` Plot the profile likelihood parameter evolution of a PROB scan. Sect. 6.3.

`--fix 'g=1.7,r_dk=-0.09'` Fix given scan parameters to the given values.

`--group LHCb` Change the GAMMACOMBO logo to LHCb.

`--id 57` When making controlplots (`--controlplots`), only plot toys generated at a specific scan point, e.g. 57. The id number refers to the scan step and therefor to a specific value of the parameter of interest.

`--importance` Activate importance sampling for PLUGIN toys. This reduces the number of toys generated in a region with large expected p value, therefore saving computing cycles, that can better be invested to get a better p value accuracy in the tails.

`--intprob` Use the internal (PROB) χ^2 histogram instead of the χ^2 from the toy files to evaluate $1 - \mathrm{CL}$ of the PLUGIN method. Sect. 6.3.

`--largest` Report largest CL interval: from the lowest boundary of all intervals to the highest boundary of all intervals. Useful if two intervals are very close together.

`--leg` Give print options for the legend such as its position, or to turn it off.

`--lightfiles` Produce only light weight PLUGIN toy files. They cannot be used for control plots but save disk space.

`--log` make logarithmic one-dimensional 1-CL plots

`--magnetic` switches on magnetic plot boundaries, that will drag the contours towards them. In many cases, this results in nicer contours, but not always. Sect. 4.3.

`--ncontours` Plot this many sigma contours in 2D plots (max 5).

`--ndivy` Set the number of axis divisions (y axis in 1D and 2D plots).

`--ndiv` Set the number of axis divisions (x axis in 1D and 2D plots).

`--nosyst` Ignore systematic uncertainties.

`--npoints2d 70` use 70 scan points in each direction, so $70^2 = 4900$ in total.

`--npoints2dx` Configure the number of scan points on x-axis in 2D scans.

`--npoints2dy` Configure the number of scan points on y-axis in 2D scans.

`--npointstoy` Number of scan points used by the plugin method.

`--npoints` Configure the number of scan points used in 1D and 2D scans (for 2D, use same number of points for each axis).

`--nrun 1` run job number 1. This number will be added to the filenames of the produced PLUGIN toy files. Files will be overwritten if the same number is given.

`--ntoys 100` produce 100 PLUGIN toys per scan point.

`--parfile` Give a specific parameter file to provide, for example, start parameters. Sect. 6.1.

`--plotid` Make the control plot with given ID to save time (`--controlplots`). Sect. 6.3.

`--pluginplotrange` Restrict the PLUGIN plot to a given range to rejcet low-statistics outliers.

`--po` make a PLUGIN-only plot.

`--prelim` Plot "Preliminiary" into the plots. See also `--unoff`.

`--printcor` Print correlation matrix of each solution found.

`--probforce` Use a stronger minimum finding algorithm for the PROB method. Sect. 6.

`--pr` physical range. This will enforce the physical parameter ranges defined in `src/ParametersTutorial.cpp`. Sect. 4.7.

`--ps 1` print a marker at all local minima. If `--ps 2` is given, only the best fit value is indicated.

`--ps 1` print solution on the plot. The given value configures the position of the numerical value on the plot. See the help `bin/tutorial -h` for options.

`--pulls` Make a pull plot illustrating the consistency of the best solution with the observables. Sect. 6.3.

`--qh` A list of quick hacks that can be enabled through the command line, that modify very specific areas of the code. These areas can be found by searching for calls to the `isQuickhack()` function.

`--scanforce, -f` Use a stronger minimum finding method for the PLUGIN method.

`--scanrange -1.5:2.5` Adjust the scan range to the given range. The default range is defined in `src/ParametersTutorial.cpp`. Sect. 4.7.

`--scanrangey -1.5:2.5` For 2D plots: Adjust the scan range of the vertical axis to the given range. The default range is defined in `src/ParametersTutorial.cpp`. Sect. 4.7.

`--sn2d` 2D version of `--sn`.

`--sn` Save nuisances to a parameter cache file at certain scan point. Sect. 6.1.

`--title` Override the title of a combination.

`--unoff` Plot "Unofficial" into the plots. See also `--prelim`.

`--var a_gaus --var b_gauss` specify the two variables to perform the scan for. The variable given first will be plotted on the horizontal axis, the second one on the vertical axis.

`--var` the variable name of the theory parameter to be scanned.

`-a plot` activates the plot action: Whenever a scan is done, the result is saved into a file in `plots/scanner`. This can be read in to remake the plot, without having to rerun the entire scan. With more complex combinations, this saves time.

`-a pluginbatch` activates the PLUGIN batch mode. This will run a number of PLUGIN toys and save them into an output file. Many of these jobs can be run in parallel for example on a batch farm.

`-a plugin` activates the PLUGIN mode. This will read in a number of PLUGIN toy files that where previously produced. These files need to be specified using the `-j` option.

`-c 1:+2` modifying combination number 1 by adding PDF number 2. Sect. 4.5.

`-c 5` combination number 5. This refers to pre-defined combinations. A list of available IDs can be obtained by running with the usage flag, `bin/tutorial -u`. When the combination argument is given multiple times, all combinations will be computed and added to the same plot. The order controls the order in the plot, where the last one is in the foreground. Sect. 4.1.

`-d` Activate debug level output.

`-i` interactive mode. Any plot will be shown directly in the familiar ROOT canvas. Exit with `Ctrl+c`.

`-j 1-3` read in PLUGIN toy files number 1 through 3.

`-u` Prints usage information and exits.

`-v` Activate verbose output.

8 Acknowledgements

The author would like to thank Max Schlupp, Marco Gersabeck, Omer Tzuk, Matt Kenzie, and Florian Bernlochner for valuable discussions and contributions to GammaCombo.

References

[1] LHCb collaboration, R. Aaij *et al.*, *Measurement of the CKM angle γ from a combination of $B^\pm \to Dh^\pm$ analyses*, Phys. Lett. **B726** (2013) 151, arXiv:1305.2050.

[2] Particle Data Group, J. Beringer *et al.*, *Review of Particle Physics (RPP)*, Phys. Rev. **D86** (2012) 010001.

[3] G. J. Feldman and R. D. Cousins, *A Unified approach to the classical statistical analysis of small signals*, Phys. Rev. **D57** (1998) 3873, arXiv:physics/9711021.

[4] B. Sen, M. Walker, and M. Woodroofe, *On the unified method with nuisance parameters*, Statistica Sinica **19** (2009) 301.

[5] T. M. Karbach, *Feldman-Cousins Confidence Levels - Toy MC Method*, arXiv:1109.0714.

[6] LHCb collaboration, R. Aaij *et al.*, *Measurement of the CKM angle γ using $B^\pm \to DK^\pm$ with $D \to K^0_S\pi^+\pi^-, K^0_S K^+ K^-$ decays*, arXiv:1408.2748.