

# Combination code (CoCo) for averaging a set of measurements

July 4, 2026

Peter Clarke<sup>1</sup>, Veronika Chobanova<sup>2</sup>

<sup>1</sup>*University of Edinburgh, Edinburgh, United Kingdom*

<sup>2</sup>*Instituto Galego de Física de Altas Enerxías (IGFAE), Universidade de Santiago de Compostela, Santiago de Compostela, Spain*

## 1 Introduction

This is a generic averaging code to combine an arbitrary collection of (experimental) measurements. It allows the inclusion of the statistical errors and their correlations, as well as systematic errors and their correlations. Correlations are handled both between the parameters of one set of measurements and between different sets of measurements. A brief explanation of the averaging method is given in Sect. ?? followed by a guide of an example code in Sect. ??.

## 2 Averaging method

In this section, details of the general averaging method for any set of measurements is presented.

### 2.1 *ResultSets*

The output of an analysis (or experiment), or an already combined set of analyses from an experiment, is referred to as a *ResultSet*. A *ResultSet* is stored in a `json` file, and contains a set of parameter values, their statistical-errors and correlation matrix, plus any systematic errors and their corresponding correlation matrices, if they exist. Example `json` files are provided in `tutorial/inputs/`.

The parameters are provided as a list of dictionaries with a name ("`Name`"), a central value ("`Value`") and an error ("`Error`", typically statistical). A statistical correlation matrix between the parameters is provided through a nested list as "`StatisticalCorrelationMatrix`". Note that the order of the individual correlations must follow the order in the "`Parameter`" list. Next, systematic uncertainties and possible correlations can be provided as a list "`SystematicErrors`". Again, the order of uncertainties and correlations must follow that of the "`Parameter`" list.

An arbitrary set of these *ResultSets*, organised in a *ResultList*, are used as the input to the averaging program. Each can contain any number of parameters. The *ResultSets* are not required to contain the same parameters. It is also possible to average parameters from different *ResultSets* which can be transformed into each other (see Sect. ??).

## 2.2 General averaging procedure

First, a vector of measurement values,  $\bar{V}_{all}^{meas}$ , is created from the set of *ResultSets* A, B, C...

$$\bar{V}_{all}^{meas} = \begin{bmatrix} \bar{V}_A \\ \bar{V}_B \\ \bar{V}_C \\ \dots \end{bmatrix}.$$

where  $\bar{V}_A$  are the measurement values of all parameters in *ResultSet* A ...etc.

Next an overall covariance matrix,  $E_{all}$ , which is block-diagonal in the *ResultSets*, is created from the errors and correlation matrices in each *ResultSet*.

$$E_{all} = \begin{bmatrix} E_A & & & \dots \\ & E_B & & \dots \\ & & E_C & \dots \\ \dots & \dots & \dots & \dots \end{bmatrix},$$

where  $E_A$  is the square matrix with dimension equal to the number of parameters measured in *ResultSet* A. This includes the statistical errors on the parameters with a correlation matrix provided in the `json` file as "Error" in the "Parameter" list and as "StatisticalCorrelationMatrix", respectively. Correlations between systematic uncertainties are taken into account as "SystematicCorrelationMatrix" for each entry in the "SystematicErrors" list. If no correlation matrix is provided, the systematic errors are considered uncorrelated and the resulting covariance matrix is diagonal.

Next, it is possible to correlate uncertainties between *ResultSets*, referred to as *inter-correlations*. These are added as off-block-diagonal sub-matrices  $C_{IJ}$  ( $I \neq J$ ).

$$E_{all} = \begin{bmatrix} E_A & C_{AB} & C_{AC} & \dots \\ C_{AB} & E_B & C_{BC} & \dots \\ C_{AC} & C_{BC} & E_C & \dots \\ \dots & \dots & \dots & \dots \end{bmatrix}.$$

The way of defining how the  $C_{AB}$  are formed based upon a supplementary set of `json` files is described in Section ??

Next, a desired set of combined output parameters,  $\bar{P}^{fit}$ , is defined, where  $\bar{P}^{fit}$  either matches parameters found in  $\bar{V}_{all}^{meas}$  or can be transformed into parameters in  $\bar{V}_{all}$  (see Sect. ??). From  $\bar{P}^{fit}$ , a vector of fit parameters,  $\bar{V}_{all}^{fit}$ , is created corresponding to each of the measured parameters,  $\bar{V}_{all}^{meas}$ .

Finally, a difference vector is created,

$$\bar{\Delta} = \bar{V}_{all}^{meas} - \bar{V}_{all}^{fit}, \quad (1)$$

and a  $\chi^2$  is constructed according to

$$\chi^2 = \bar{\Delta}^T E_{all}^{-1} \bar{\Delta}. \quad (2)$$

The resulting  $\chi^2$  is minimised with respect to  $\bar{P}^{fit}$  using `Minuit` [?], determining a set of parameters, uncertainties and an overall correlation matrix. These can be used as input for future averages.

## 2.3 Correlations between systematic uncertainties in different *ResultSets*

Inter-correlations between each common systematic uncertainty in two or more *ResultSets* are much harder to take account of in a fully generic way. For example one could specify each and every correlation in some `json` file with thousands of entries. This was not considered to be useful.

We have chosen instead to implement a set of pre-defined maps. Currently two types of maps are implemented:

1. *"Diagonal-only"*: if parameter  $X$  appears in *ResultSets* A and B, the systematic errors in two sets are assumed to be  $s\%$  correlated in  $C_{AB}$  between  $X_A$  and  $X_B$  but otherwise no correlation between different parameters  $X_A$  and  $Y_B$ , ( $X \neq Y$ ). The default is  $s = 100\%$ . We have also allowed for the case that  $X_A$  and  $X_B$  may be effectively the same parameter for this purpose, but not actually have exactly the same names - we include a parameter equivalence feature to handle this specified in the `json` files.
2. *"Full"*: for each pair of parameters  $X_A$  and  $Y_B$  in *ResultSets* A and B respectively, a search is performed for correlations  $\rho_{XY}$  declared within both A and B. If correlations are found in both A and B, the smaller correlation is taken as the correlation between  $X_A$  and  $Y_B$  in  $C_{AB}$ . If no such correlation is found in either A or B then no correlation is assumed. Again an arbitrary scale factor can be applied that is 100% by default. Note that this automatically includes the *"Diagonal-only"* case above. The parameter equivalence feature is also included.

The type of inter-correlation for each systematic uncertainty must be provided in a `json` file(s), see an example in `tutorial/inputs/InterCorrelationMaps-All.json`. The name of the inter-correlated systematic uncertainties must be the same in all *ResultSets* and it must be specified whether the inter-correlation is *"Diagonal-only"* ("DIAG") or *"Full"* ("FULL").

The `json` file formally applies to inter-correlations between single pair of *ResultSets*, whose labels appear in the files. However a "special" label="All" is recognised, which causes the file to be applied across all *ResultSets*. Thus, an arbitrary number of these files can be provided to specify limited inter-correlations between subsets of *ResultSets* if required. However for simple use only a single file specifying label="All" is needed.

The file also contains lists of parameters which might have different names in different *ResultSets*, but which are considered as equivalent (i.e. effectively the same parameter) for the purposes of inter-correlation. An example might be  $\Gamma$  in one set and  $\Gamma_s$  in another. This is implemented via a set of *"ParameterEquivalenceLists"* included in the `json` files. An example is discussed in Sect. ??.

## 2.4 Controlling treatment of errors

It was found useful to be able to dynamically control how errors are treated in the average (for testing purposes). The program allows the user to set flags to easily switch on and off: (i) inclusion of Statistical Correlations, (ii) inclusion of Systematic Errors or not, (iii) inclusion of Systematic Correlation Matrices or not and (iv) whether inter-correlations are included or not.

## 2.5 Translators

In some cases, different measurements of the same property would use different parametrisations of the parameters. Such is for instance a measurement of a particle lifetime,  $\tau$ , which can also be measured as a decay width,  $\Gamma$ , instead. The two parameters are related via  $\Gamma = \hbar/\tau$ . In such cases, the parameters can be averaged using *Translators*, which are functions acting on the parameters provided by the user as a separate module (see `src/combination_code/TranslatorsExample.py`). These functions transform the parameters into one another allowing to combine them into one parameter. The transformation considers automatically the change in the uncertainties and correlations of the parameter with respect to the rest of the parameters.

It is obviously not possible to provide a pre-coded set of all possible translators. The user may therefore expect to provide additions to this module as required.

### 3 Example code

We provide an example of a combination of four independent measurements (*ResultSets*) in `tutorial/Example4` which is executed as

```
$ cd tutorial/Example4
$ python CombineResultsExample.py
```

The example combines four measurements stored in two `json` files in `tutorial/inputs/`, `ResultList-Example-JpsiKK.json` and `ResultList-Example-JpsiPiPi.json`. These are based on results published in Refs. [?, ?, ?, ?] with example correlation matrices provided for illustration.

A description of the functions within a module (e.g. `HelperFunctions`, `IntercorrelationMaps`, `TranslatorsExample`) can be obtained by running

```
$ pydoc <ModuleName>
```

In this example, the four measurements contain the parameters:

- *ResultSet* "2012-JPsiKK": "gamma", "deltaGammas", "AperpSq", "AzeroSq", "para", "perp", "phis", "lamb", "dms"
- *ResultSet* "2016-JPsiKK": "gsgd", "deltaGammas", "AperpSq", "AzeroSq", "para", "perp", "phis", "lamb", "dms", "FS1", "FS2", "FS3", "FS4", "FS5", "FS6", "deltaS1", "deltaS2", "deltaS3", "deltaS4", "deltaS5", "deltaS6"
- *ResultSet* "2012-JPsiPiPi": "lambJpsiPiPi", "phis"
- *ResultSet* "2016-JPsiPiPi": "ghgd", "lambJpsiPiPi", "phis".

Note that the only common parameter between all sets is "phis". In addition, the parameters "gamma", "gsgd" and "ghgd" are equivalent as reflected in `InterCorrelationMaps-All.json`. These correspond to  $B_s^0$  and  $B_d^0$  meson decay widths and differences,  $\Gamma_s$ ,  $\Gamma_s - \Gamma_d$  and  $\Gamma_H - \Gamma_d$ , respectively. Equivalent parameters are considered as the same parameter when inter-correlating *ResultSets*. They are converted into one another through the translators defined in `TranslatorsExample.py`. Furthermore, the  $B_d^0$  meson decay width [?] is an external Gaussian constraint to these measurements, which is added through a *ResultSet* in a separate `json` file, `ResultList-Gammad.json`.

The set of independent parameters to be averaged is defined in `CombineResultsExample.py` as `pars = ['phis', 'lamb', 'lambJpsiPiPi', 'gamma', 'deltaGammas', 'dms', 'AperpSq', 'AzeroSq', 'perp', 'para', 'FS1', 'FS2', 'FS3', 'FS4', 'FS5', 'FS6', 'deltaS1', 'deltaS2', 'deltaS3', 'deltaS4', 'deltaS5', 'deltaS6', 'gammaD']`.

Note that "gamma", "gsgd" and "ghgd" are all averaged into one parameter, "gamma".

The user can choose whether to include statistical, systematic or inter-correlations in the average by setting `doStatisticalCorrelations`, `doSystematics`, `doSystematicCorrelations`, `doIntercorrelations` to `True` or `False`.

The averaged result is saved as a *ResultSet*, which can be used in a subsequent average. A pull plot of the input parameters with respect to the result as well as the fit  $\chi^2$  per degrees of freedom is provided for information.

The example code has been written so that a new user only needs to change file names and flags in one function (`main`) to get going. All output is controlled from here as well. Optional access to another function is provided (`doAverage`) to allow the user to change Minuit behaviour should they wish, but it is not necessary to modify this to get going.