

# Combination code (CoCo) tutorial

**Veronika Chobanova**

21 May 2026

# CoCo overview

*CoCo is a generic averaging code to combine an arbitrary set of (experimental) results under consideration of the statistical errors and their correlations, and systematic errors and correlations, both between the parameters of one measurement and different measurements.*

- Code is python based
- Written by Peter Clarke and Veronika Chobanova, prompted by averages for LHC(b) measurements of  $\phi_s$ 
  - drives to some extent examples and available features

CoCo provides averaging features such as

- multidimensional averages
- correlations between measurements and parameters
- correlations between single systematic uncertainties
- conversion between related parameters
- usage of output of one average as an input to another...

HFLAV averages in CoCo

- $\Delta m_s$ : verified against COMBOS
- $\tau(B_s^0)$ ,  $\Delta\Gamma_s$  and  $\phi_s$ : verified against previous code

# Goals of this tutorial

- Explain the functioning principles of CoCo
- Provide a guide to the most important features
- Give a hands-on experience through example averages

# Getting started

- Create a python3 virtual environment:  
`python3 -m venv coco-env`
- Activate the virtual environment:  
`cd coco-env`  
`source bin/activate`
- Install the code with dependencies:  
`$ python -m pip install combination-code`
- Documentation and examples are available in the `share/combination-code` directory
- Run example to verify it works  
`$ cd tutorial/Example4`  
`$ python CombineResultsExample.py`
- If you see a folder outputs with json, tex etc files, you are ready to go

# How CoCo works

CoCo uses as inputs ResultSets in json consisting of *central values*, *errors* and *correlation matrices*

```
{
  "ResultSet": [
    {
      "ResultSetLabel": "CDF 9.6 fb$^{ -1}$",
      "Description": [
        "CDF, Phys.Rev.Lett 109,171802 (2012), http://journals.aps.org/prl/pdf/10.1103/PhysRevLett.109.171802"
      ],
      "Parameter": [
        {
          "Name": "DGs",
          "Value": 0.068,
          "Error": 0.026
        },
        {
          "Name": "phis",
          "Value": -0.24,
          "Error": 0.36
        }
      ],
      "StatisticalCorrelationMatrix": [
        [ 1.00 , 0.0 ],
        [ 0.0 , 1.00 ]
      ],
      "SystematicErrors": [
        {
          "Name": "TotalSyst",
          "Values": [0.009, 0.0],
          "SystematicCorrelationMatrix":
            [[ 1.00 , 0.0 ],
             [ 0.0 , 1.00 ]]
        }
      ]
    }
  ],
}
```

# How CoCo works

- As a first step, define a desired set of output parameters,  $\vec{P}^{\text{fit}}$
- Any average boils down to minimizing a  $\chi^2$  wrt  $\vec{P}^{\text{fit}}$  constructed as

$$\chi^2 = \Delta^T E^{-1} \Delta \quad (1)$$

- $E^{-1}$ : overall covariance matrix
- $\Delta = \vec{V}^{\text{meas}} - \vec{V}^{\text{fit}}$ : a vector of differences between input central values,  $\vec{V}^{\text{meas}}$ , and fit parameters,  $\vec{V}^{\text{fit}}$
- $\vec{V}^{\text{meas}}$  either matches parameters found in or can be transformed into parameters in  $\vec{P}^{\text{fit}}$
- $\vec{V}^{\text{fit}}$  is a vector of fit parameters created from  $\vec{P}^{\text{fit}}$ , with the same length as  $\vec{V}^{\text{meas}}$
- Output same format as inputs, can be directly used in another average

# How CoCo works

- For the case of measurements A, B, C..., with parameters  $\vec{V}_A$ ,  $\vec{V}_B$ ,  $\vec{V}_C$ ... and overall covariance matrices (include statistical and systematic uncertainties)  $E_A$ ,  $E_B$ ,  $E_C$ ...

$$\vec{V}_{meas} = \begin{bmatrix} \vec{V}_A \\ \vec{V}_B \\ \vec{V}_C \\ \dots \end{bmatrix} \quad E = \begin{bmatrix} E_A & & & \dots \\ & E_B & & \dots \\ & & E_C & \dots \\ \dots & \dots & \dots & \dots \end{bmatrix}$$

- It is also possible to correlate systematic uncertainties between measurements by providing a recipe to construct covariance matrices between measurements,  $C_{AB}$ ,  $C_{AC}$ ,  $C_{BC}$ ..., in which case

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



# CoCo code structure

- **Base code** consists of three files with a total of  $< 1400$  lines of code organized in three scripts
  - `ResultSet.py` is the core of the code where the two main classes are defined, `ResultSet` and `ResultSetList`
  - `HelperFunctions.py` provide mainly functions for printouts of results to files and in the terminal
  - `IntercorrelationMaps.py` handles correlations between measurements (to be discussed later in more details)
- **The user** needs to provide
  - An **averaging script** to define inputs, parameters, averaging function etc. Example provided: `tutorial/Example4/CombineResultsExample.py`  
Further examples in this tutorial, see `tutorial/Example1,2,3`
  - A **set of results to average** organized as `ResultSets` in json files, see for example `tutorial/inputs/ResultList-Example-JpsiKK.json`
  - (Optional) An **intercorrelation map** in case of correlating measurements
  - (Optional) A **set of translators** in case of averaging related parameters

## Example 1: Average of $CP$ -even $\tau(B_s^0)$

- Simplest case of a 1D average from two measurements without correlations
- To run

```
$ cd combination-code/tutorial/Example1
$ python Example1Average.py
```
- Note both inputs in the same file organized in a ResultList,  
`../inputs/ResultList-tau-CPeven-Spring2020.json`

Simple or naive average same as Minuit  
result as no correlations considered

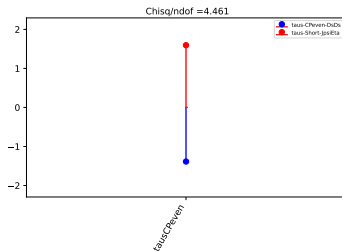
Pull plot to visualise inputs compared  
to average

```
IMinuit average
tausCPeven      :      1.4220  +/-      0.0234

IMinuit MINOS average
tausCPeven      :      1.4220  +  0.0234  -0.0234

Simple Average of ResultList:
tausCPeven      :      1.4220  +-      0.0234

Final Corrmat:
1.00
nInputs = 2 nFitParams= 1
chi2 = 4.461 nDoF = 1.0 chi2/nDoF = 4.461 p-value = 0.03468
```



## Example 1: Average of $CP$ -even $\tau(B_s^0)$

- Result output saved under outputs/tauCPEven.json
- Same format as inputs, note StatisticalCorrelationMatrix in this case means the full correlation matrix that comes out of the fit

```
{
  "ResultSet": [
    {
      "ResultSetLabel": "CombinationOutputResultSet",
      "Description": [ "CombinationOutputResultSet" ],
      "Parameter": [
        {
          "Name": "tausCPEven",
          "Value": 1.422035,
          "Error": 0.023443
        }
      ],
      "StatisticalCorrelationMatrix": [
        [ 1.00 ]
      ]
    }
  ]
}
```

# Correlating measurements

- To correlate measurements, the user needs to provide a json file with details

```
{  
  "ResultSetLabels" : ["All", "All" ],  
  
  "IntercorrelationMaps": [  
    {"lhcb_pzscales"      : {"ictype": "DIAG", "scale":1.0, "strategy": "AVG" }}  
  ],  
  "ParameterEquivalenceLists" : [  
    ""  
  ]  
}
```

- "ResultSetLabels": Specifies which results to intercorrelate.  
Need two entries at least, if "All" specified, will correlate all measurements
- "ParameterEquivalenceLists": To specify parameters that are named differently but are equivalent for averaging purposes

# Correlating measurements

- "IntercorrelationMaps": Dictionary with names of systematics to correlate and related attributes
  - scale: To scale correlations, mainly for testing. Otherwise set to 1
  - ictype: Possible options include DIAG and FULL, relevant for multidimensional averages.

DIAG: Correlate only same/equivalent parameters between measurements, i.e. covariance matrices between measurements, e.g.  $C_{AB}$ ,  $C_{BC}$  would be diagonal

FULL: Correlate all parameters between measurements, including different parameters, i.e. covariance matrices between measurements could contain non-zero off-diagonal elements

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

# Correlating measurements

- "IntercorrelationMaps": Dictionary with names of systematics to correlate and related attributes
  - strategy: Strategy to calculate correlation between two parameters (1 and 2) from two ResultSets (A and B) in the FULL case. Possible options include
    - When correlations between the two parameters are available in both data sets
$$\begin{aligned}(\text{SET})\text{MIN} &= \min(|\rho_{12}^A|, |\rho_{12}^B|) \cdot (\text{sign}(\min(|\rho_{12}^A|, |\rho_{12}^B|))) \\ (\text{SET})\text{MAX} &= \max(|\rho_{12}^A|, |\rho_{12}^B|) \cdot (\text{sign}(\max(|\rho_{12}^A|, |\rho_{12}^B|))) \\ (\text{SET})\text{AVG} &= (\rho_{12}^A + \rho_{12}^B)/2 \\ (\text{SET})\text{SQRT} &= \begin{cases} \text{sign}(\rho_{12}^A) \cdot \sqrt{\rho_{12}^A \cdot \rho_{12}^B}, & \text{if } \text{sign}(\rho_{12}^A) = \text{sign}(\rho_{12}^B) \\ \text{AVG}, & \text{otherwise} \end{cases}\end{aligned}$$
    - When a correlation  $c$  between the two parameters is available only in one of the two data sets
      - $\text{SET}\{\text{MIN}, \text{MAX}, \text{AVG}, \text{SQRT}\} = c$

## Example 2: Average of $\Delta m_s$

- A simple 1D average with correlated systematic uncertainties
- Correlating "pzcales" in all LHCb results

### From input json file

```
{
  "ResultSetLabel": "LHCb-Ds-mu+X",
  "Description": [
    "LHCb Ds-mu+X 1fb-1"
  ],
  "Parameter": [
    {
      "Name": "dms",
      "Value": 17.93,
      "Error": 0.22
    }
  ],
  "StatisticalCorrelationMatrix": [
    [ 1.00 ]
  ],
  "SystematicErrors": [
    {
      "Name": "lhcb_pzcales",
      "Values": [0.03]
    },
    {
      "Name": "RestSyst",
      "Values": [0.149331845]
    }
  ]
},
```

### Inter correlations json file

```
{
  "ResultSetLabels": ["All", "All"],
  "InterCorrelationMaps": [
    {"lhcb_pzcales": {"ictype": "DIAG", "scale": 1.0, "strategy": "AVG"}}
  ],
  "ParameterEquivalenceLists": [
    ""
  ]
}
```

### Set up in averaging script

```
#This is where you configure your list of input JSON files to configure any inter correlations (optional)
interCorrelationFileList = [
  'InterCorrelationMaps-Dms.json'
]

#Show whats in the ResultList of ResultSets
reslist = readResultListJSON(inputfile, inputpath='', icfilelist=interCorrelationFileList)

reslist.setErrorTreatmentFlags( doStatCorr=True, doSyst=True, doSystCorr=True, doInterCorr=True )
```

## Example 2: Average of $\Delta m_s$

- Informative printout on correlations and measurements considered

```
InterrelationMaps: rs1=LHCb-Ds-3pi+ rs2=LHCb-Ds-mu+X : applied diagonal intercorr for: lhcb_pzscales
InterrelationMaps: rs1=LHCb-Ds-3pi+ rs2=LHCb-Ds-pi+ : applied diagonal intercorr for: lhcb_pzscales
InterrelationMaps: rs1=LHCb-Ds-mu+X rs2=LHCb-Ds-pi+ : applied diagonal intercorr for: lhcb_pzscales
```

- Note simple average clearly different from fit result with correlations

```
IMinuit average
dms          :      17.7642 +/-      0.0229

IMinuit MINOS average
dms          :      17.7642 + 0.0229 -0.0229

Simple Average of ResultList:
dms          :      17.7636 +-      0.0227
```



# Averaging related parameters

- CoCo provides a functionality to average related parameters
- E.g. if experiment A provides a measurements of a lifetime parameter  $\Gamma^A$  and experiment B provides  $\tau^B$ , one can average them to a common  $\Gamma$  or  $\tau$
- To do this, you need to provide a Translator
- Translators are functions that 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 wrt the rest of the parameters  
→ no need to manually convert parameters into one another beforehand

## Example 3: Usage of translators and multidimensional averages

- Example 3 is a simplified average of the  $B_s^0 \rightarrow J/\psi K K$  measurements by the LHC
- The experiments quote different sets of parameters which can still be combined
- E.g. CMS and LHCb quote the polarisation fractions  $|A_{\perp}|^2$  and  $|A_0|^2$  and ATLAS quotes  $|A_{\parallel}|^2$  and  $|A_0|^2$   
→ fit  $|A_{\perp}|^2$  in the combination transforming  $|A_{\parallel}|^2 = 1 - |A_0|^2 - |A_{\perp}|^2$

```
{ ATLAS
  "Name": "AparaSq",
  "Value": 0.2220,
  "Error": 0.0027
},
```

```
{ CMS
  "Name": "AperpSq",
  "Value": 0.2393,
  "Error": 0.0062
},
```

```
{ LHCb
  "Name": "AperpSq",
  "Value": 0.2489,
  "Error": 0.0035
},
```

## Example 3: Usage of translators and multidimensional averages

### Including translators

- Add to module Translators.py
- Call them in the averaging function
  - need to provide a dictionary with fit parameter names and values

### Translator for $|A_{||}|^2$ in Translators.py

```
def AparaSqTranslator( fitDict ):  
    #Calculates AparaSq in terms of the fit parameters  
    try:  
        AperpSq = fitDict[ 'AperpSq' ]  
        AzeroSq = fitDict[ 'AzeroSq' ]  
        AparaSq = 1-AperpSq-AzeroSq  
        return AparaSq  
    except:  
        print(' \n Translator for AparaSq called, but it cant find parameters it can use - exiting \n')  
        sys.exit()
```

### Adding translators to averaging script

```
#Add all translators. They must appear in the Translators.py module  
AddAllTranslators( reslist )
```

### Fit parameters - note no $|A_{||}|^2$

```
parameters = ["AzeroSq", "AperpSq", "AsSqCMS", "para", "perp", "SperpCMS", "Gs", "DGs", "dms", "lamb", "phis", "Gd"]
```

## Example 3: Usage of translators and multidimensional averages

- Another use case of translators is to produce averages of related parameters
- E.g. easily do  $(\tau_s, \Delta\Gamma_s/\Gamma_s)$  and  $(\tau_{sH}, \tau_{sL})$  from  $(\Gamma_s, \Delta\Gamma_s)$

```
parametersRat = ["AzeroSq", "AperpSq", "AsSqCMS", "para", "perp", "SperpCMS", "taus", "DGs/Gs", "dms", "lamb", "phis", "Gd"]  
m, resphisGsRat = doAverage( reslist, parametersRat, "WriteUp2021JpsiKKRata.json", fix_dms=17.757)
```

```
parametersLH = ["AzeroSq", "AperpSq", "AsSqCMS", "para", "perp", "SperpCMS", "tausL", "tausH", "dms", "lamb", "phis", "Gd"]  
m, resphisGsLH = doAverage( reslist, parametersLH, "WriteUp2021JpsiKKLHa.json", fix_dms=17.757)
```

# Final remarks

## Other features

- External constraints: Just add a `ResultSet` with the central value and error on the parameter(s) you want to constrain
- `printLatexTable`: Prints final correlation table in a LaTeX file

## CoCo maintenance

- Extensions are possible and straight forward to implement as code based on `python`
- Feedback and suggestions always welcome
- New contributions also welcome

Thank you!