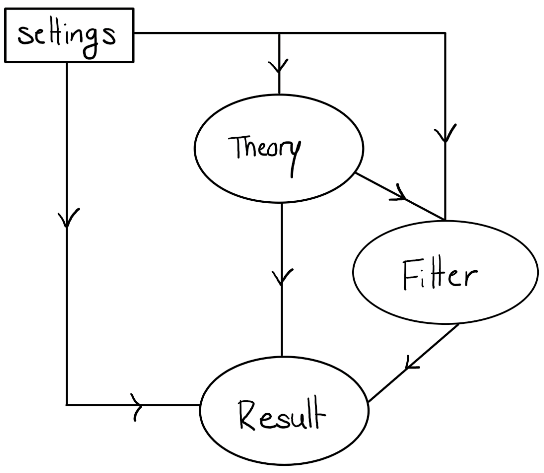
Notes on PySIMBA

Stella Hoffmann
s08shoff@uni-bonn.de

1 Introduction

This is a short overview on the python project pysimba. The original code is written in c++ and has some more features, which can still be implemented in pysimba. This code can run the fit on given data and plot the results. The fit results give the same values as the c++ results, which was the first main goal of this project. If now new measurements are added to the code, you should be able to change some things in the settings and then run it just like before. In the following I'll explain how to do that.

2 Structure



The code is based on 3 main classes (+ Settings and Tools):

- **Theory:** Provides the fit-function using the given leading and subleading theory.
- **Fitter:** Provides a function to execute the fit using `iminuit` and a Fitter object containing the results of the fit.
- **Result:** Visualizes the Fitter results.

In the settings everything around the fitting process will be set, so the code itself must not be changed for doing several fits. The options are described below.

2.1 Some functions

2.1.1 Theory class

```
    Theory.Chisq(self, par: Vector)->float
    >>> type Vector = list[float]
```

Calculates χ^2. This function will be minimized in the `Fitter.DoSingleFit(...)`-function.

2.1.2 Fitter class

```
     Fitter.DoSingleFit(self, NumbPar: int = 3,
        with_minos: bool = True)

     \\ Example Usage:
     >>> import py_simba
     >>> fit_obj = py_simba.Fitter()
     >>> fit_obj.DoSingleFit(NumbPar = 4)
```

This function will conduct a single fit with the amount of `NumbPar`-parameters. Which measurements should be included in this single fit, can be chosen in the settings. The fitter object itself stores the poperties of the fit:

```
    Fitter.m: Minuit     // Minuit object after the fit
    Fitter.mb: float     // mass mb
    Fitter.chisq: float // mimimal value for chisq
    Fitter.theo: Theory // Fitted Theory obj
    Fitter.NumbOfPar: int
    Fitter.Lambda: float
```

2.1.3 Result class

```
    Result.CalculatePrediction(self, key: str, fit:
       Fitter)
```

Calculates the prediction for a specific measurement with a given fit.

2.2 Input

2.2.1 Settings in .yml file

Variable	Explanation	Options
Tag	Tag for the name of the produced files	any name
ResultPath	Path where the results should be stored	any path
TheoryOrder	Strings for leading theory which should be used in the calculation of the prediction. (mid's)	`NNLLNNLO,NLLNLO` are options, rest is nessecary
SubLeadTheoryOrder	One string for the used subleading theory	`'SSF27_'+subleading_end`
TheoryTag	Which functional form is used	Options: `[expx3, expx4, gaussx4]`
SubLeadingTheoryTag		Only option yet `'SSF27'`
TheoryPath	Path of used theory dictionary, depending on uses functional form	any path to theory dictionary
SubleadingTheoryPath	Path to the subleading theory dictionary	any path
TheoryMomentsPath	Path to dictionary with moments, same for every functional form because of `TheoryTag`	any path

Table 1: Settings overview 1

Variable	Explanation	Options
MeasurementPath	Path to measurement dictionary	any path
KeyOrder	list of strings with measurement keys that should be included in the fit	`['belle', 'babar_hadtag', 'babar_incl', 'babar_sem']`
Minimum	If not every value should be included put here the first index that should be included for each measurement	0−`max`
Scale	If the measurement should be scaled	any number
BasisExpansion	Depending on the λ	every leading_end
FitVars	Naming of the fit parameters, put a list of strings in there	any list of names
StartValues	List of start values, should be as long as `FitVars`	some values that make sense
NumbPar	Integer of how many parameters one wants to fit	any integer
Constants	Definition of used constants	
SubLeadCoefficients	Used in calculation of `SubLeadPars()`	Default value is 0 ($d_2 = 0$)

Table 2: Settings overview 2

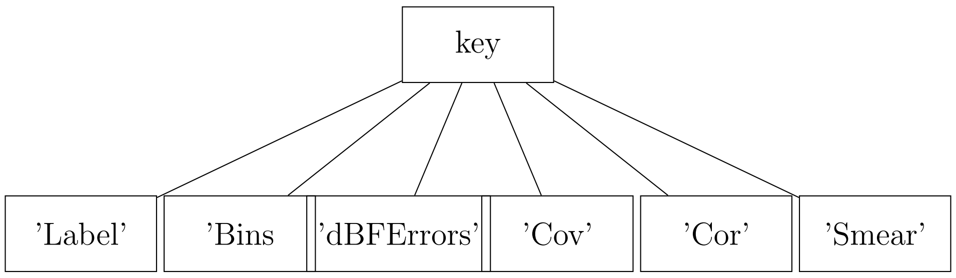
2.2.2 Dictionaries

The structure of each used dictionary is as follows. Please note, that everything written in 'Quotation marks' is a keyword to access the data of that dictionary. If it's not written with quotation marks, then it is a variable with changeable input. Here are the possibilities for the used variables:

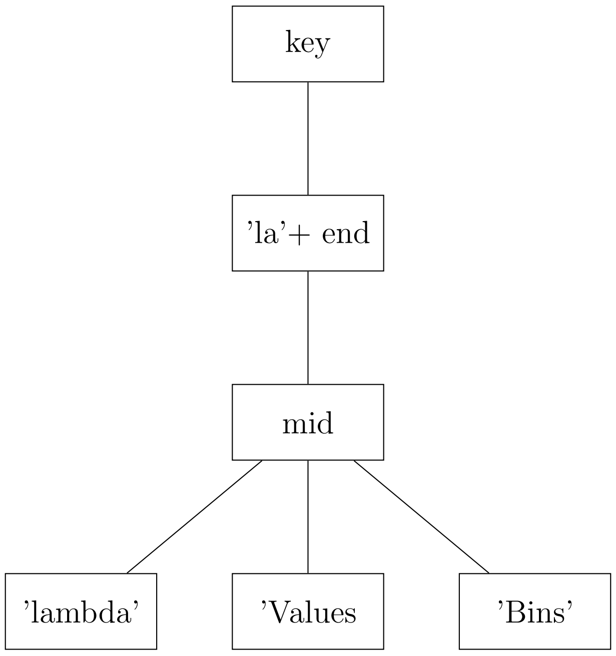
key: { 'belle', 'babar_hadtag', 'babar_incl', 'babar_sem' } [1]
mid: { 'NNLLNNLO', 'NS22NNLO', 'NS27NNLO', 'NS28NNLO', 'NS78NNLO', 'NS88NNLO'} [2]
end(for leading theory): { '03', '04', '05', '06', '07', '08', '09', '10', '11', '12', '035', '045', '055', '065', '075', '085', '095', '0475', '0525', '0575' , '0625'}
end(for subleading theory): { '105', '105', '107', '205', '206', '207', '1045', '1055', '1065', '2045', '2055', '2065', '10525', '10575', '10625', '20525', '20575'}

[1] ATTENTION: 'babar_sem' is currently not working with the fit. Input still has to be checked.
[2] Also known as elements of *TheoryOrder*

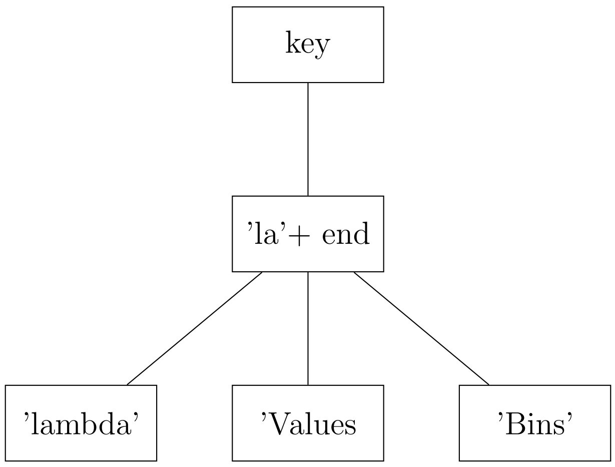
Measurement



Leading Theory



Subleading Theory



With these dictionaries every data the python code needs can be accessed. They are stored in pickle files (.pkl). The path to those pickles can be set in the config file (*settings.yml*).

2.3 Installing and Running

Installing
If your shell is located in the PySimba Directory, which you can clone you can install the package locally with:
`pip install -e .`
Or the path to the package instead of '.'

Usage
If you run it in the python terminal:

```
import py_simba
py_simba.main()
```

Or just test other functions alone.

In the `py_simba.main()` there will be two options to choose from. Either run the fit or add a new measurement. If you choose the second option you must have access to the `PySIMBA/src/py_simba/data/add/` directory, where you must place the dictionaries as pickle files which you want to add to the ones above. If you did this you can run the second option and the dictionaries which you choose will be expanded.

Be careful because this changes your input dictionairies!!!

If you want to run it just as a package in your shell:
`python3 -m py_simba`

If you want to use different functions, for example the χ^2 function, do it as follows:

```
>>> import py_simba
>>> import numpy as np
>>> theo_obj = py_simba.Theory()
>>> par = np.array([0.1,0.1,0.1])
>>> py_simba.Theory.Chisq(theo_obj, par)
np.float64(259085.2755991831)
```

3 How to add a new measurement

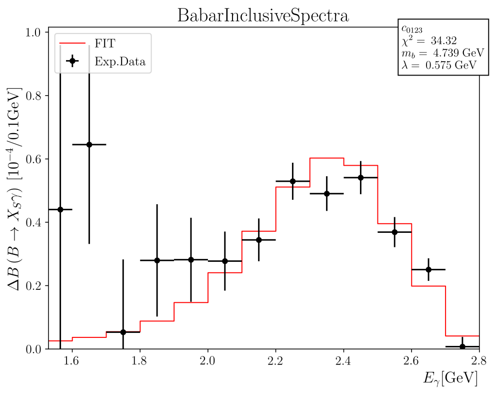
1. Add data to the dictionaries (Measurement, Leading Theory, Subleading Theory)
   - Come up with a `key` for your measurement and add the dictionary to the others:
     - Add your dictionary to the `PySIMBA/src/py_simba/data/add/` directory
     - Run `py_simba.main()` and choose the second option to Add a new measurement
     - Follow the instructions in the shell. When the programm wants to know the name of the dictionary which is located in `PySIMBA/src/py_simba/data/add/new_dict.pkl`, only type `new_dict`.
     - After that you have to choose a key for the new measurement this can be anything that defines your measurement for example `'belle2'`.
     - After that the measurement should be added to the dictionaries. It tells you the new key's in the used dictionary, you should find your key there
   - everything should now be accessible through the key of your measurement, when you structured it as described in the previous chapter 2.2.2.
2. Add your `key` to the settings file, if you want to include it in the fit. The setting `KeyOrder` should be changed so it looks somewhat like this `['belle', 'babar_hadtag', 'babar_incl', 'babar_sem', 'your_new_key']`

3. Run the code and choose option '1' :)

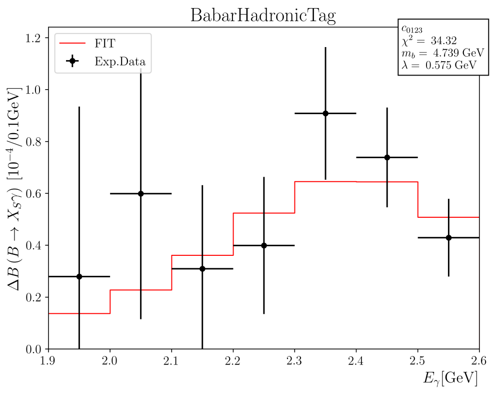
4 Results

The `Run()` function will produce a `result.txt`-file, which contains $\chi^2, m_b, norm, a_n$'s. It will also produce the plots for the fit you have run and put them in the `result`-directory as well.
Here are some examples:

```
Results for 4 Parameters
Chisq = 34.3231
mb = 4.7389
norm = 0.00501
a1 = 0.04943
a2 = 0.09100
a3 = -0.01569
```



(a) Babar inclusive spectra



(b) Babar hadronic tag

Figure 1: Fit for 4 Parameters using 3 measurements.

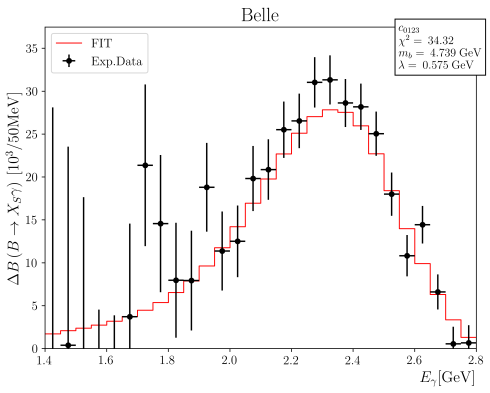


Figure 2: Belle