EOS — A HEP Programm for Flavor Physics

User Manual

Danny van Dyk
Christoph Bobeth
Frederik Beaujean

version btoplxnu-9-g0ba60

August 7, 2016

Contents

Acknowledgments

The creation and maintenance of EOS would not have been possible without the assistance and encouragement of Gudrun Hiller.

Furthermore, we would like to extend our thanks to the following people whose input and support were most helpful either the development or the maintenance of EOS, either through personal contributions to the code, independent code review, or helpful suggestions: Thomas Blake, Christoph Langenbruch, David Leverton, Ciaran McCreesh, Hideki Miyake, Konstantinos Petridis, Denis Rosenthal, Alexander Shires, Ismo Toijala, Christian Wacker .

Part I.

Documentation

1. Installation

1.1. Installing the Dependencies

Installing EOS from source will require the following software:

g++ the GNU C++ compiler, in version 4.8.1 or higher,

autoconf the GNU tool for creating configure scripts, in version 2.69 or higher,

automake the GNU tool for creating makefiles, in version 1.14.1 or higher,

libtool the GNU tool for generic library support, in version 2.4.2 or higher,

pkg-config the freedesktop.org library helper, in version 0.28 or higher.

Building and using the core libraries requires in addition the following software:

GSL the GNU Scientific Library [1], in version 1.16 or higher,

HDF5 the Hierarchical Data Format v5 library [2], in version 1.8.11 or higher,

Minuit2 the physics analysis tool for function minimization, in version 5.28.00 or higher.

Except for **Minuit2**, we recommend the installation of the above packages using your system's software management system. For **Minuit2**, we recommend to install from source, and to disable the automatic support for OpenMP. The installation can be done using:

```
mkdir /tmp/Minuit2
pushd /tmp/Minuit2
wget http://www.cern.ch/mathlibs/sw/5_28_00/Minuit2/Minuit2-5.28.00.tar.gz
tar zxf Minuit2-5.28.00.tar.gz
pushd Minuit2-5.28.00
./configure --prefix=/opt/pkgs/Minuit2-5.28.00 --disable-openmp
make all
sudo make install
popd
popd
rm -R /tmp/Minuit2
```

If you intend to use the Population Monte Carlo (PMC) sampling algorithm, you will need to install

libpmc a free implementation of said algorithm [3], in version 1.01 or higher,

in addition to the core library dependencies. The **libpmc** package will – in all likelihood – not be installable via your system's software management system. In addition, EOS requires some modifications to libpmc's source code, in order to make it compatible with C++. We suggest the following commands to install it:

```
mkdir /tmp/libpmc
pushd /tmp/libpmc
wget http://www2.iap.fr/users/kilbinge/CosmoPMC/pmclib_v1.01.tar.gz
tar zxf pmclib_v1.01.tar.gz
pushd pmclib_v1.01
./waf configure --m64 --prefix=/opt/pkgs/pmclib-1.01
```

```
./waf
sudo ./waf install
sudo find /opt/pkgs/pmclib-1.01/include -name "*.h" \
    -exec sed -i \
    -e 's/#include "errorlist\.h"/#include <pmctools\/errorlist.h>/' \
    -e 's/#include "io\.h"/#include <pmctools\/io.h>/' \
    -e 's/#include "mvdens\.h"/#include <pmctools\/mvdens.h>/' \
    -e 's/#include "maths\.h"/#include <pmctools\/maths.h>/' \
    -e 's/#include "maths_base\.h"/#include <pmctools\/maths_base.h>/' \
    {} \;
sudo sed -i \
    -e 's/^double fmin(double/\/\/&/' \
    -e 's/^double fmax(double/\/\/&/' \
    /opt/pkgs/pmclib-1.01/include/pmctools/maths.h
popd
popd
rm -R /tmp/pmclib
```

1.2. Installing EOS

The most recent version of EOS is contained in the public GIT [4] repository at `http://github.com/eos/eos`. In order to download it for the first time, create a new local clone of said repository via:

```
git clone \
    -o eos \
    -b master \
    https://github.com/eos/eos.git \
    eos
```

As a first step, you need to create all the necessary build scripts via:

```
cd eos
./autogen.bash
```

Next, you configure the build scripts using:

```
./configure \
    --prefix=/opt/pkgs/eos \
    --with-minuit2=/opt/pkgs/Minuit2-5.28.00 \
    --with-pmc=no
    # --with-pmc=/opt/pkgs/pmclib-1.01
```

Note that the last line is optional, and should replace the second to last line only if you intend to use the PMC sampling algorithm. If the `configure` script finds any problems with your system, it will complain loudly.

After successful configuration, you can build and install EOS using:

```
make all
sudo make install
```

In order to be able to use the EOS clients from the command line, you will need to

```
export PATH+=":/opt/pkgs/eos/bin"
```

to you `.bashrc` file. In order to build you own programs that use the EOS libraries, add

```
CXXFLAGS+=" -I/opt/pkgs/eos/include"
LDFLAGS+=" -L/opt/pkgs/eos/lib"
```

to your makefile.

Moreover, we urgently recommend to also run the complete test suite upon installation, using:

```
make check
```

within the source directory. Please contact the authors in the case that any test failures should occur.

2. Usage

EOS has been authored with two use cases in mind.

The first such use case is the evaluation of observables and further theoretical quantities in the field of flavor physics. EOS aspires to produce such evaluations in the course of theory estimates of publication quality.
The second use case is the inference of parameters from experimental observations. For this task, EOS defaults to the Bayesian framework of parameter inference.

In the remainder of this chapter, we document the usage of the existing EOS clients and scripts, in order to carry out tasks corresponding to the above use cases. We assume further that only the built-in observables, physics models and experimental constraints are used.

2.1. Evaluating Observables using `eos-evaluate`

Observables can be evaluated using the `eos-evaluate` client. It accepts the following command line arguments:

`--kinematics NAME VALUE`

Within the scope of the next observable, declare a new kinematic variable with name `NAME` and numerical value `VALUE`.

`--range NAME MIN MAX POINTS`

Within the scope of the next observable, declare a new kinematic variable with name `NAME`. Subdivide the interval [MIN, MAX] in POINTS subintervals, and evaluate the observable at each subinterval boundary.

Note: More than one `--range` command can be issued per observable, but only one `--range` command per kinematic variable.

`--observable NAME`

Add a new observable with name `NAME` to the list of observables that shall be evaluated. All previously issued `--kinematics` and `--range` arguments apply, and will be used by the new obervable. The kinematics will be reset (i.e., all kinematic variables will be removed) in anticipation of the next `--observable` argument.

The output of calls to `eos-evaluate` is structured as follows:

- The first row names the observable at hand, as well as all active options.
- The second row contains column headers in the order:
  - kinematics variables,
  - the upper and lower uncertainty estimates for each individual uncertainty budget,

  - the total upper and lower uncertainty estimates.

- The third row contains the result as described by the above columns. In addition, at the end of the row the relative total uncertainties are given in parantheses.

The above structure repeats itself for every observable, as well as for each variation point of the kinematic variables as described by occuring `--range` arguments.

As an example, we turn to the evaluation of the q^2-integrated branching ratio $\mathcal{B}(\bar{B}^0 \to \pi^+\ell^-\bar{\nu}_\ell)$. For this example, let us use the integration range

$$0\,\mathrm{GeV}^2 \leq q^2 \leq 12\,\mathrm{GeV}^2\,.$$

Further, let us use the BCL2008 [5] parametrization of the $\bar{B} \to \pi$ form factor, as well as the Wolfenstein parametrization of the CKM matrix. The latter is achieved by choosing the physics model 'SM'. By default, EOS uses the most recent results of the UTfit collaboration's fit of the CKM Wolfenstein parameter to data on tree-level decays. In this example, we will evaluate the observable, and estimate parametric uncertainties based on the naive expectation of Gaussian uncertainty propagation. We will classify two budgets of parametric uncertainties: one for uncertainties pertaining to the form factors (labelled 'FF'), and one for uncertainties pertaining to the CKM matrix elements (labelled 'CKM').

Our intentations translate to the following call to `eos-evaluate`:

```
eos-evaluate \
    --kinematics s_min  0.0 \
    --kinematics s_max 12.0 \
    --observable "B->pilnu::BR,l=e,form-factors=BCL2008" \
    --budget "FF" \
    --vary "B->pi::f_+(0)@BCL2008" \
    --vary "B->pi::b_+^1@BCL2008" \
    --vary "B->pi::b_+^2@BCL2008" \
    --budget "CKM" \
    --vary "CKM::lambda" \
    --vary "CKM::A" \
    --vary "CKM::rhobar" \
    --vary "CKM::etabar"
```

2.2. Producing Random Parameter Samples

Both use cases, observable evaluation and Bayesian parameter inference, make use of random samples of some Probability Density Function (PDF) $P(\vec{\theta})$. These random samples can be produced from Markov chains, using the Metropolis-Hastings algorithm, by calls to the `eos-sample-mcmc` client. In second step, refined samples, or samples for a very complicated setup, are obtained from an algorithm described in Ref. [6]. This algorithm uses an adaptive importance sampling called Population Monte Carlo (PMC), implemented within the client `eos-sample-pmc`.

The `eos-sample-mcmc` clients accepts the following command-line arguments:

`--scan NAME --prior flat MIN MAX`

`--scan NAME [ABSMIN ABSMAX] --prior gaussian MIN CENTRAL MAX`

`--nuisance [...]`

These arguments add a parameter to the statistical analysis, with either a flat or a gaussian prior. If `ABSMIN` and `ABSMAX` are specified, the prior will be cropped to this absolute interval. The `--scan` and `--nuisance` arguments work identically, with one exception: `--nuisance` declares the associated parameter as a nuisance parameter, which is flagged in the HDF5 output. The sampling algorithm *does not* treat nuisance parameters differently than scan parameter.

`--seed [time|VALUE]`

This argument sets the seed value for the Random Number Generator (RNG). Setting the seed to a fixed numerical `VALUE` ensures reproducibility of the results. This is important for publication-quality usage of the client. If `time` is specified, the RNG is seeded with an interger value based on the current time.

`--prerun-min VALUE`

For the prerun phase of the sampling algorithm, set the minimum number of steps to `VALUE`.

`--prerun-max`

For the prerun phase of the sampling algorithm, set the maximum number of steps to `VALUE`.

`--prerun-update`

For the prerun phase of the sampling algorithm, force an adaptation of the Markov chain's proposal function to its environment after every `VALUE` steps.

`--store-prerun [0|1]`

Either disable or enable storing of the prerun samples to the output file.

Note: Samples from the prerun should only be used for diagnostic purpose.

`--output FILENAME`

Use the file `FILENAME` to store the output, using the HDF5 file format.

The `eos-sample-pmc` clients accepts the following command-line arguments:

`--scan NAME --prior flat MIN MAX`

`--scan NAME [ABSMIN ABSMAX] --prior gaussian MIN CENTRAL MAX`

`--nuisance [...]`

These arguments add a parameter to the statistical analysis, with either a flat or a gaussian prior. If `ABSMIN` and `ABSMAX` are specified, the prior will be cropped to this absolute interval. The `--scan` and `--nuisance` arguments work identically, with one exception: `--nuisance` declares the associated parameter as a nuisance parameter, which is flagged in the HDF5 output. The sampling algorithm *does not* treat nuisance parameters differently than scan parameter.

`--seed [time|VALUE]`

This argument sets the seed value for the RNG. Setting the seed to a fixed numerical `VALUE` ensures reproducibility of the results. This is important for publication-quality usage of the client. If `time` is specified, the RNG is seeded with an interger value based on the current time.

`--pmc-initialize-from-file HDF5FILE`

Use the samples from a MCMC HDF5 output file `HDF5FILE` as generated with `eos-sample-mcmc`, in order to initialize the mixture density of the initial PMC step.

`--pmc-group-by-r-value R`

When forming groups of MCs from the initialization file, only add a chain to an existing group if the chain's R-value is less than `R`; create a new group otherwise.

`--hc-target-ncomponents N`

When creating mixture components, create `N` components per existing MC group.

`--hc-patch-length LENGTH`

When clustering a group's MCs onto the mixture components, cut the chains into patches of `LENGTH` samples each.

`--hc-skip-initial FRACTION`

Skip the first `FRACTION` of all MCMC samples in the clustering step.

Note: `FRACTION` must be a floating point number between 0 and 1.

`--pmc-samples-per-component N`

Set the number `N` of samples that will be drawn per component and update step of the PMC run.

`--pmc-final-samples N`

Set the number `N` of samples that will be drawn for the final step, i.e.: after the PMC updates have converged.

`--pmc-ignore-ess [0|1]`

Set whether convergence of the PMC updates shall be determined from the effective sample size (ESS) *in addition* to the perplexity.

Default: Use the ESS.

`--pmc-relative-std-deviation-over-last-step STD STEPS`

If both perplexity and ESS have a standard deviation less than `STD` over the last `STEPS` updates, declare convergence.

`--output FILENAME`

Use the file `FILENAME` to store the output, using the HDF5 file format.

As an example, we define the a-priori PDF for a study of the decay $\bar{B} \to \pi^+\ell^-\bar{\nu}_\ell$. For the CKM Wolfenstein parameters, we use

$$\begin{aligned} \lambda &= 0.22535 \pm 0.00065\,, & A &= 0.807 \pm 0.020\,, \\ \bar{\rho} &= 0.128 \pm 0.055\,, & \bar{\eta} &= 0.375 \pm 0.060\,. \end{aligned}$$

For the a-priori PDF, we use uniform distributions for the BCL2008 [5] parameters. However, we construct a likelihood from the results of a recent study of the form factor $f_+^{B\pi}(q^2)$ within Light-Cone Sum Rules (LCSRs). We now intend to draw random numbers from the posterior PDF using EOS' adaptive Metropolis-Hasting algorithm. During its prerun phase, the algorithm adapts the chains' proposal functions. We should demand at least 500, and – for a problem of this complexity – maximally 7500 steps during the prerun phase; the adaption process should be executed after every 500 steps. For the final samples, we wish for a total of 10^4, which we artifically decompose into 10 chunks with 1000 samples each.

Our intentions translate to the following call to `eos-sample-mcmc`:

```
eos-sample-mcmc \
    --global-option model CKMScan \
    --global-option form-factors BCL2008 \
    --scan "CKM::abs(V_ub)"          2e-3   5e-3  --prior flat \
    --scan "B->pi::f_+(0)@BCL2008"   0      1     --prior flat \
    --scan "B->pi::b_+^1@BCL2008"  -20    +20     --prior flat \
    --scan "B->pi::b_+^2@BCL2008"  -20    +20     --prior flat \
    --constraint "B->pi::f_+@IKMvD-2014" \
    --constraint "B^0->pi^+lnu::BR@BaBar-2010B" \
    --constraint "B^0->pi^+lnu::BR@Belle-2010A" \
    --constraint "B^0->pi^+lnu::BR@BaBar-2012D" \
    --constraint "B^0->pi^+lnu::BR@Belle-2013A" \
    --prerun-min     500 \
    --prerun-max    7500 \
    --prerun-update  500 \
    --prerun-only        \
    --chunks          10 \
    --chunk-size    1000 \
    --output /tmp/mcmc_prerun_btopi+ff.hdf5
```

We use the above call to `eos-sample-mcmc` in order to initialize a PMC run. We wish for 4 components per MC group, and to skip 20% of the MCMC samples as part of the burn in. Groups will be grouped based on an R-value threshold of 1.5. For each update, 500 samples per mixture component shall be drawn, in order to produce 10^6 samples in the final step. Convergence shall be declared upon a standard deviation for the perplexity only, of 0.05 over the last 4 update steps. The call then reads:

```
eos-sample-pmc \
    --global-option model CKMScan \
    --global-option form-factors BCL2008 \
    --scan "CKM::abs(V_ub)"          2e-3   5e-3  --prior flat \
    --scan "B->pi::f_+(0)@BCL2008"   0      1     --prior flat \
    --scan "B->pi::b_+^1@BCL2008"  -20    +20     --prior flat \
    --scan "B->pi::b_+^2@BCL2008"  -20    +20     --prior flat \
    --constraint "B->pi::f_+@IKMvD-2014" \
    --constraint "B^0->pi^+lnu::BR@BaBar-2010B" \
    --constraint "B^0->pi^+lnu::BR@Belle-2010A" \
    --constraint "B^0->pi^+lnu::BR@BaBar-2012D" \
    --constraint "B^0->pi^+lnu::BR@Belle-2013A" \
    --pmc-initialize-from-file /tmp/mcmc_prerun_btopi+ff.hdf5 \
    --hc-target-ncomponents 4 \
    --hc-skip-initial 0.2 \
    --pmc-samples-per-component 500 \
    --pmc-group-by-r-value 1.5 \
    --pmc-final-samples 1000000 \
    --pmc-ignore-ess true \
    --pmc-relative-std-deviation-over-last-steps 0.05 4 \
    --output /tmp/pmc_monolithic_btopi+ff.hdf5
```

2.3. Plotting Random Parameter Samples

Once random samples from the posterior PDF have been obtained, e.g. as described in section **??**, a visual inspection of the samples is the next step. EOS provides scripts for this purpose, which plot histograms of either a marginalized 1D (`eos-plot-1d`) or 2D (`eos-plot-2d`) probability density. Both scripts presently detect the output format by inspection of the resepective HDF5 file name. Files containing MCMC samples should be prefixed with `mcmc_`, while PMC sample files should be prefixed with `pmc_monolithic_`.

The `eos-plot-1d` produces a 1D histogram of the samples for one parameter. It accepts the following arguments:

`HDF5FILE`

The name of the HDF5 input file whose samples shall be plotted.

`IDX`

The numerical index for the parameter whose density function shall be plotted. `IDX` starts with zero

`PDFFILE`

The name of the PDF output file, into which the plot shall be saved.

`--xmin XMIN,--xmax XMAX`

When specified, limit the plot range to the interval `XMIN` to `XMAX`. The default values are taken from the description of the parameter within the HDF5 input file.

`--kde [0|1]`

When enabled, plots a univariate Kernel Density Estimate of the probability density based on the available samples.

`--kde-bandwidth KDEBANDWIDTH`

When specified, multiplies the automatically determined KDE bandwidth parameter with `KDEBANDWIDTH`.

The `eos-plot-2d` produces a 2D heatmap of the samples for two parameters. It accepts the following arguments

`HDF5FILE`

The name of the HDF5 input file whose samples shall be plotted.

`XIDX`

The numerical index for the parameter that shall be plotted on the x acis. `XIDX` starts with zero

`YIDX`

The numerical index for the parameter that shall be plotted on the y acis. `YIDX` starts with zero

`PDFFILE`

The name of the PDF output file, into which the plot shall be saved.

`--xmin XMIN,--xmax XMAX`

When specified, limit the plot range on the x axis to the interval `XMIN` to `XMAX`. The default values are taken from the description of the parameter within the HDF5 input file.

`--ymin YMIN,--ymax YMAX`

When specified, limit the plot range on the y axis to the interval `YMIN` to `YMAX`. The default values are taken from the description of the parameter within the HDF5 input file.

3. Library Interface

We begin this chapter by explaining the rationale behind several of the design decisions of the EOS libraries in section 3.1. Subsequently, we document the core set of C++ classes in section 3.2.

3.1. Design

In order to fulfill its intended use cases, the EOS libraries are designed with concepts in mind.

First, most of the scalar quantities that used within the EOS libraries are treated as parameters. These start with directy experimental input, such as particle masses and lifetimes. They continue along the lines of more theoretically motivated quantities, such as quark masses (in the $\overline{\text{MS}}$ scheme) and the Wolfenstein parameters of the CKM matrix. It is therefore straightforward to change a parameter's role within the scope of a theoretical analysis, from being a nuisance parameter in the course of producing some estimates to being a genuine parameter of interest in the course of a fit. In order to differentiate between the various parameters, a naming scheme is put in place. Within this scheme, a parameter's name is rendered:

`NAMESPACE::ID@TAG,` (3.1)

where the meta variables take the following meaning:

NAMESPACE A short description of the context in which the parameter should be interpreted. Possible namespaces include, but are not limited to: `mass`, `decay-constants`, `CKM`, and others.

ID A handle for a parameter in a given namespace.

TAG Usually a reference, which allows to distinguish between parameters with the same `ID`.

Second, if a quantity exhibits a functional dependence on either a parameter or a kinematic variable, it is treated in a modular fashion. Per default, an abstract interface is created that allows for several implementation of the same quantity. The actual implementations of this interface are then accesible via a factory method. An excellent example for such a *plugin* design is found within the scope of hadronic matrix elements, in particular hadronic form factors; see e.g. section **??**. Each implementation of a plugin usually depends on its own set of parameters. For clarity, we use one of the hadronic form factor as an example. Consider the $B \to \pi$ vector form factor $f_+^{B\pi}$. One possible parametrization of this form factor has been dubbed BCL2008 [5]. It is achieved in terms of three parameters: the normalization $f_+^{B\pi}(0)$, and two shape parameters $b_1^{B\pi,+}$ and $b_2^{B\pi,+}$. This parametrization is implemented within EOS, andobservables that depend on the $B \to \pi$ form factors can choose it through the option `form-factors=BCL2008`. The relevant parameters are contained in the namespace `B->pi`, and tagged for the BCL2008 plugin:

`B->pi::f_+(0)BCL2008,` `B->pi::b_+1̂BCL2008,` and `B->pi::b_+2̂BCL2008.` (3.2)

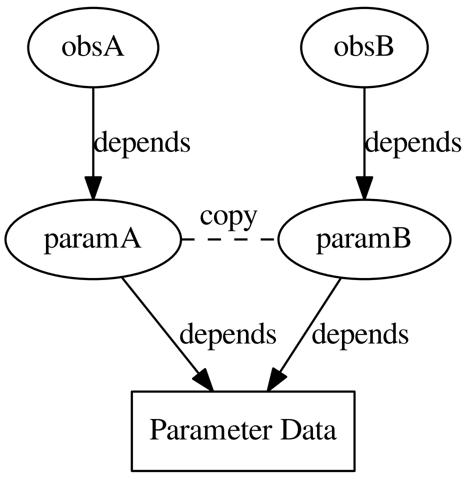


Figure 3.1.: Diagrammatic illustration that multiple observables can depend on the very same instance of `Parameters`.

As such, there is no risk of namespace collisions among the various plugins' parameters.

3.2. Core Classes

At the core of the EOS API there are four classes – `Parameters`, `Kinematics`, `Options`, `Observable` – all of which are discussed in the following.

3.2.1. Class **`Parameters`**

`key = value` dictionary, with string keys and floating-point real values

- copies share, by default, the parameters of the original
- observables usually share a common set of parameters

```
Parameters paramA = Parameters::Defaults();
Parameters paramB(paramA);

ObservablePtr obsA = Observable::make("A", paramA, Kinematics{ }, Options{ });
ObservablePtr obsB = Observable::make("A", paramB, Kinematics{ }, Options{ });
```

- access to individual parameters via array subscript `[]`
  - input: parameter name
  - result: instance of `Parameter`, w/ persistent access to parameter data
    lookup once, use often!

- parameter naming scheme: `NAMESPACE::ID@SOURCE`, e.g.:
  - `mass::b(MSbar)` $\rightarrow$ mass $\overline{m}_b(\overline{m}_b)$ in $\overline{\text{MS}}$ scheme
  - `B->K::f_+(0)@KMPW2010` $\rightarrow$ normalization of f_+ FF in $B \rightarrow K$ decays, according to KMPW2010

3.2.2. Class `Kinematics`

The class `Kinematics` is a dictionary from `std::string`-valued keys to `double`-valued entries. Upon construction of an observable, a suitable instance of kinematics is bound to that observable.
`key = value` dictionary, with string keys and floating-point real values

- allows run-time construction of observables
- each obervable has its very own set of kinematic variables
- access to individual variables via array subscript `[]`
  - input: variable name
  - result: double
- no naming scheme, since namespace is unique per observable instance

3.2.3. Class `Options`

`key = value` dictionary, with string keys and string values influences how observables are evaluated

- access to individual otions via array subscript `[]`
  - input: option name
  - result: string value
- example: lepton flavour in semileptonic decay:
  `l=mu, l=tau,...`
- example: choice of form factors:
  `form-factors=KMPW2010 ...`
- example: `model=...` as choice of underlying physics model

  `SM` to produce SM prediction

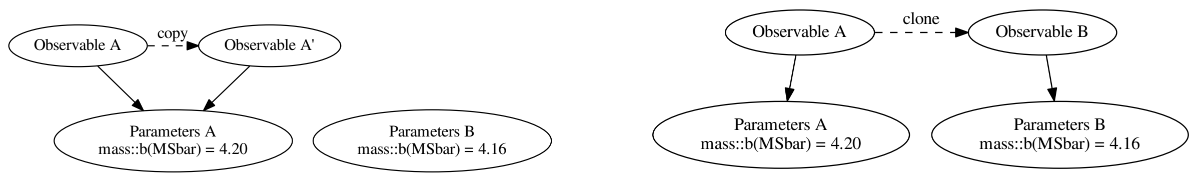
  `WilsonScan` to fit Wilson coefficients

  `CKMScan` to fit CKM matrix elements

3.2.4. Class `Observable`

`Observable` is an abstract base class

- Its descendants must at construction time:
  - associate with an instance of Parameters



(a) copy

(b) clone

Figure 3.2.: Diagrammatic illustration of the differences between copying an instance of `Observable` via the copy-constructor, and cloning the observable via the `clone` method

  - extract values from their instance of Options

- Their construction occurs via factory method, e.g.:
  `Observable::make("B->pilnu::BR", p, k, o)`

  The latter creates an observable at runtime using the observable's name (here: `B->pilnu::BR`), a set of parameters `p`, a set of kinematic variables `k`, and a set of options `o`.

- Any changes to the observable's instance `p` of `Parameters` after construction of the observable transparently affect the observable, and all further observable associated with `p`.

- Any changes to `o` of `Options` do not affect the observable after construction time.

- Observables can be
  - evaluated via `evaluate`:

    This methods runs the necessary computations for the present values of the parameters

  - copied:

    The copy-constructor does not create an independent copy; the new object rather uses the same parameters, with the same options as the original.

  - cloned:

    The method `clone` creates an independent copy of the same observable, using a different set of parameters than the original

- All users of `Observable` must also support cloning. This allows to easily parallelize algorithms within EOS.

Acronyms

LCSR Light-Cone Sum Rule. 8

PDF Probability Density Function. 6, 8

PMC Population Monte Carlo. 3, 4

RNG Random Number Generator. 7

Bibliography

[1] J. T. M. Galassi et al., *GNU Scientific Library*.

[2] The HDF Group, *The HDF5 library and data format*.

[3] M. Kilbinger et al., *CosmoPMC*.

[4] L. T. Junio C. Hamano, Shawn O. Pearce et al., *GIT*.

[5] C. Bourrely, I. Caprini, and L. Lellouch, "Model-independent description of $B \to \pi\ell\bar{\nu}$ decays and a determination of $|V_{ub}|$", Phys.Rev. **D79**, 013008 (2009).

[6] F. Beaujean, "A bayesian analysis of rare b decays with advanced monte carlo methods", PhD thesis (Fakultät für Physik, Technische Universität München, 2012).