EOS — A HEP Program for Flavour Physics

User Manual

Danny van Dyk
Christoph Bobeth
Frederik Beaujean

version XXX

August 31, 2016

Contents

Acknowledgments

Software Documentation

1. Installation

The aim of this section is to assist you in the installation of EOS from source. For the remainder of the section, we will assume that you build and install EOS on a Linux based operating system, such as Debian or Ubuntu.[1] Installation on MacOS X is known to work, but not guaranteed to work out of the box.

1.1. Installing the Dependencies

The dependencies can be roughly categorized as either system software, or scientific software.

System software Installing EOS from source will require the following system software to be pre-installed:

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.

If you intend to use the Python [1] interface to the EOS library, you will need to additionally install

libboost-python the BOOST library for interfacing Python and C++.

We recommend that you install the above packages only via your system's software management system.

Scientific Software Building and using the EOS core libraries requires in addition the following scientific software to be pre-installed:

GSL the GNU Scientific Library [2], in version 1.16 or higher,

HDF5 the Hierarchical Data Format v5 library [3], 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.[2] For **Minuit2**, we recommend to install the package from source, and to disable the automatic support for OpenMP. The installation can be done by executing the following commands:

[1]Other flavours of Linux will work as well, however, note that we will exclusively use package names as they appear in the Debian/Ubuntu apt databases.
[2]There is presently a bug in the Debian/Ubuntu packages for **Minuit2**, which prevents linking.

```
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 with EOS, you will need to install

libpmc a free implementation of said algorithm [4], 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 [5] 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 \
    --enable-python \
    --with-minuit2=/opt/pkgs/Minuit2-5.28.00 \
    --with-pmc=no
    # alternative 1: --disable-python
    # alternative 2: --with-minuit2=root
    # alternative 3: --with-pmc=/opt/pkgs/pmclib-1.01
```

In the above, three alternatives apply:

1. Building the EOS-Python interface is purely optional. In order to disable building this interface, replace `--enable-python` with `--disable-python`.
2. If you have installed ROOT on your system, you can use ROOT's internal copy of Minuit2. In such a case, issue `--with-minuit2=root` to the configure script.
3. Building the EOS-PMC support is purely optional. In order to enable building this support, issue `--with-pmc` with the appropriate installation path.

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
```

Moreover, we urgently recommend to also run the complete test suite by executing:

```
make check
```

within the source directory. Please contact the authors in the case that any test failures should occur. In order to be able to use the EOS clients from the command line, you will need to set up some environment variables. For the Bash, which is the default Debian/Ubuntu shell this can be achieved by adding the lines

```
export PATH+=":/opt/pkgs/eos/bin"
export PYTHONPATH+=":/opt/pkgs/eos/lib/python2.7/site-packages"
```

to you `.bashrc` file. In the above, the last line is optional and should only be added if you built EOS with Python support. Note that `python2.7` should be replaced by the apppropriate Python version against which EOS was built.
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.

Python 3 If you intend to build the EOS-Python interface using Python 3, there will be additional steps required on Debian/Ubuntu. You will need to pass the environment variables `PYTHON` and `BOOST_PYTHON_SUFFIX` to the configure script, e.g. like this:

```
PYTHON=python3 BOOST_PYTHON_SUFFIX=-py34 ./configure \
    ...
```

where the dots indicate all the options passed to `configure` on the commandline. Note that is is a bad idea to simultaneously install the EOS-Python interface for both Python 2.x and Python 3.x, since the `PYTHONPATH` environment variable is used by both versions and there is no versioning support for Python modules written in C++.

2. Usage

EOS has been authored with several 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 theory estimates of publication quality, and has produced such estimates in the past.
- The second use case is the inference of parameters from experimental observations. For this task, EOS defaults to the Bayesian framework of parameter inference.
- The third use case is the production of toy events for a variety of flavor-physics-related processes.

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. The necessary steps to extend EOS with new observables, physics models or constraints will be discussed in chapter 4.

2.1. Evaluating Observables

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.

`--vary NAME`

Estimate the uncertainty based on variation of the parameter `NAME`, as if the parameter was distributed like a univariate Gaussian.

`--budget NAME`

Create a new uncertainty budget, which encompasses all the subsequently issued `--vary` commands (until the issue of a new `--budget` command). By default, the `delta` budget always exists, and encompasses *all* variations.

As an example, we turn to the evaluation of the q^2-integrated branching ratio $\mathcal{B}(\bar{B}^0 \to \pi^+\mu^-\bar{\nu}_\mu)$, which can be addressed in EOS through the observable name `B->pilnu::BR`. For this example, let us use the integration range

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

Further, let us use the BCL2008 [6] 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. Here, 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 intentions 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=mu,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"
```

The above call yields the following output:

```
# B->pilnu::BR: form-factors=BCL2008,l=mu
# s_min s_max central FF_min FF_max CKM_min CKM_max delta_min delta_max
0 12 0.000106816 3.00927e-05 1.46426e-05 9.14007e-06 9.70515e-06 3.14501e-05 1.75669e-05   \
    (-29.4434% / +16.446%)
```

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.

2.2. Producing Random Parameter Samples

For all the previously mentioned use cases (observable evaluation, Bayesian parameter inference, and production of pseudo events) one requires to draw random samples from some arbitrary Probability Density Function (PDF) $P(\vec{\theta})$. When using EOS, these random samples can be produced from Markov-Chain random walks, using the Metropolis-Hastings algorithm, by calls to the `eos-sample-mcmc` client. In a second step, refined samples or samples for a very complicated setup, can be obtained from an algorithm described in Ref. [7]. This algorithm uses an adaptive importance sampling called Population Monte Carlo (PMC), implemented within the client `eos-sample-pmc`, and requires a prior run of `eos-sample-mcmc` for initialization.
The follow command-line arguments are common to the `eos-sample-mcmc` and `eos-sample-pmc` clients, as well as further clients described in subsequent sections:

```
--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 treats nuisance parameters *in the same way as* scan parameters.

```
--constraint NAME
```

The named constraint from the internal database will be used as part of the likelihood. The functional form of the the likelihood, details such as correlations, and the required options for the observables used will be automatically looked up. In order to browse the entries of the database, use the `eos-list-constraints` client.

```
--fix NAME VALUE
```

The value of parameter `NAME` will be set to the supplied `VALUE`, and thus potentially deviate from its default value.

The `eos-sample-mcmc` client further accepts the following arguments:

```
--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.

`--output FILENAME`

Use the file `FILENAME` to store the output, using the HDF5 file format. The resulting HDF5 file follows the EOS-MCMC format, and can be accessed using, e.g., the `eosdata` Python module.

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

`--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.

`--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 decimal number between 0 and 1.

`--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.

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

Set the number `N` of samples that will be drawn per component and per 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-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.

`--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.

`--output FILENAME`

Use the file `FILENAME` to store the output, using the HDF5 file format. The resulting HDF5 file follows the EOS-PMC structure, and can be accessed using, e.g., the `eosdata` Python module.

As an example, we define the a-priori PDF for a study of the decay $\bar{B} \to \pi^+\mu^-\bar{\nu}_\mu$. 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 [6] parameters for the $B \to \pi$ form factor $f_+^{B\pi}$. However, we construct a likelihood from the results of a recent study (IKMvD2016 [8]) 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. As a consequence, the prerun samples will in general *not* be distributed as the posterior PDF. With a subequent PMC sampling run in mind, we should demand at least 500 steps, and – for a problem of this complexity – maximally 7500 steps during the prerun phase; the adaption process should be executed after every 500 steps.

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        \
    --output /tmp/mcmc_prerun_btopi+ff.hdf5
#    --chunks          10 \
#    --chunk-size    1000 \
```

Optionally, if the prerun converges, the client can be used to perform a main run, in which the proposal functions will be kept static. For such these main run samples, we wish for a total of 10^4, which we artifically decompose into 10 chunks with 1000 samples each. While the sampling at hand will be quite quick, sampling of computationally expensive functions should be done with small chunks so that the progress of the computations can be monitored. The require options for these intentation are shown above with a leading hash mark. Also, for a main run, the `--prerun-only` flag would need to be removed.

We use the above call to `eos-sample-mcmc` in order to initialize a PMC run. We wish for 4 mixture components per MC group, and to skip 20% of the MCMC samples as part of the burn in. MC groups will be created based on an R-value [9] 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
```

Both clients output copious amounts diagnostic data to the standard output, which include

- all information about the prior and the likelihood as specified on the command line (both clients);
- information about the convergence of the Markov chains within the parameter space, based on the R-value criterion [9] (MCMC only);
- information about the convergence of the PMC run based on the perplexity and effective sampling size (PMC only).

We display the outcome of both the MCMC (prerun) sampling as well as the PMC sampling steps in figure 2.1.

2.3. Finding the Mode of a Probability Density

The mode, best-fit point, or simply the most-likely value of some PDF $P(\vec{\theta})$ is regularly searched for in physics analyses. EOS supplies the client `eos-find-mode`, which accepts the common set of arguments describing the PDF as already discussed for the `eos-sample-mcmc` and `eos-sample-pmc` clients, see section 2.2 for further information. In addition, it accepts the following command-line arguments:

`--starting-point` $\{$ `VALUE1 VALUE2 ... VALUEN` $\}$

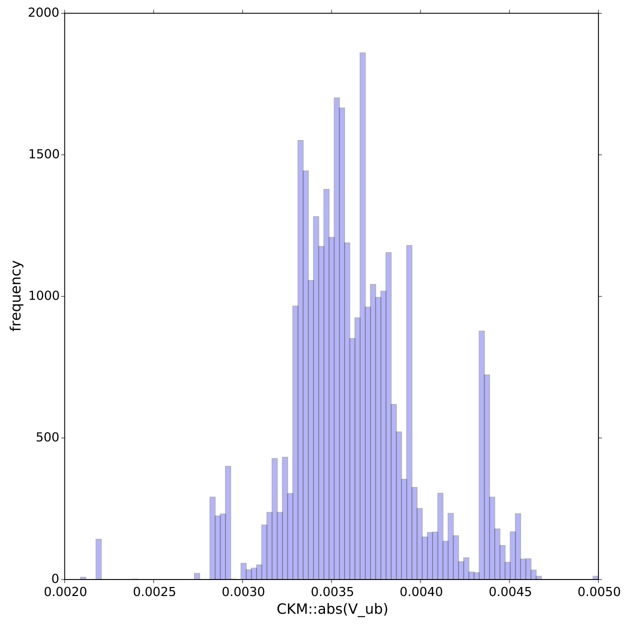
Set the starting point for the maximization of the PDF $P(\vec{\theta})$ at $\theta = (\texttt{VALUE1}, \ldots, \texttt{VALUEN})$.

`--max-iterations INTEGER`

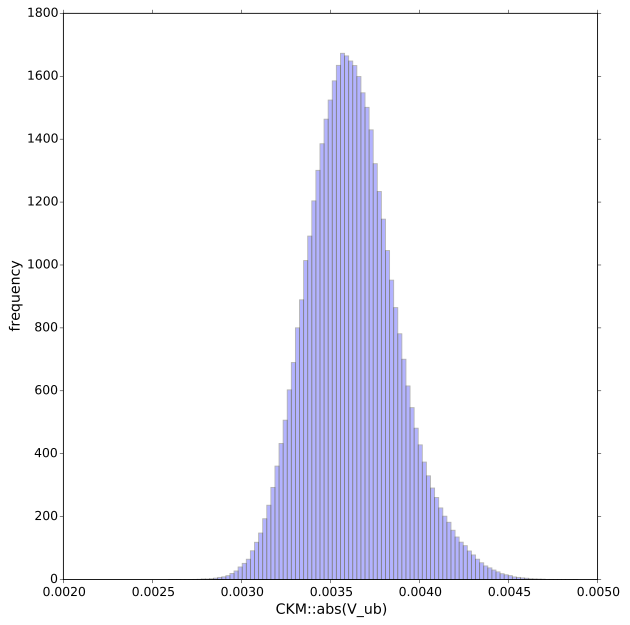
The optimization algorithm is allowed to run at maximum `NUMBER` iterations.

`--target-precision NUMBER`

Attempt to determine the mode up to an uncertainty of `NUMBER`.



(a) Histogram of the marginal posterior of $|V_{ub}|$, using 4×7500 samples. Despite the poor quality of these samples, they can be used to initialize the PMC run as described in the text.



(b) Histogram of the marginal posterior of $|V_{ub}|$, using 10^6 samples.

Figure 2.1.: Histograms of the parameter of interest $|V_{ub}|$ in the two example fits as described in section 2.2, and plotted using the `eos-plot-1d` client; see section 2.5 for details.

In order to illustrate the client's usage, we use the same example as discussed in section 2.2. The corresponding call then reads:

```
eos-find-mode \
    --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" \
    --starting-point { 3.5e-3 0.31 0 0 } \
    --max-iterations 1000 \
    --target-precision 1e-8
```

The output starts with same diagnostic information on the composition of prior and likelihood as for the sampling clients. The results are displayed in the last few lines, including

- the starting point for the mode-finding process;
- the coordinates of the best-fit point; and
- the log-posterior at the best-fit point.

```
# Starting optimization at ( 0.0035 0.31 0 0 )

# Found maximum at:
#   ( 3.568778e-03 2.661032e-01 -2.670912e+00 2.231637e-02 )
#   value = -3.101594e+02
```

2.4. Bayesian Uncertainty Propagation

EOS presently supports two ways to prograpate theory uncertainties in the framework of Bayesian statistic: First, by using prior PDF that describes the nuisance parameters; second, by using samples from a posterior PDF that have been obtained from running `eos-sample-pmc`.[1] It accepts the following command-line arguments:

`--vary NAME --prior flat MIN MAX`

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

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.

`--samples NUMBER`

Sets the number of samples that shall be produced per observable and worker thread.

`--workers NUMBER`

Sets the number of worker threads.

`--pmc-input HDF5FILE BEGIN END`

Use the samples at index `BEGIN` up to index `END` from a named data set in the file `HDF5FILE`.

Note: The file must have been produced by the `eos-sample-pmc` client.

`--pmc-sample-directory NAME`

Use the named data set within the file specified with `--pmc-input`. Valid names are `/data/X`, where X stands for either `initial`, `final`, or all numbers describing existing update steps that were carried out in the PMC run.

Note: You should use `/data/final` unless you are debuging the PMC algorithm.

As an example of the first way, we would like to predict the branching ratio for the decay $B^- \to \tau^-\bar{\nu}_\tau$. This prediction involves two parameters: First, the absolute value of V_{ub}; and second, the value of the B-meson decay constant f_B. For the former, we choose the HFAG average of the inclusive determination $|V_{ub}| = (4.45\pm0.26)\cdot 10^{-3}$ [10], while for the latter we use the FLAG average $f_B = 0.188\pm0.007\,\text{GeV}$ [11]. Our intention translates to the following call to `eos-propagate-uncertainty`:

```
eos-propagate-uncertainty \
    --global-option model CKMScan \
    --global-option form-factors BCL2008 \
    --vary "CKM::abs(V_ub)"         2e-3   5e-3  --prior gaussian 4.19e-3 4.45e-3 4.71e-3 \
    --vary "decay-constant::B_u"    0.167  0.209 --prior gaussian 0.181   0.188   0.195   \
    --observable "B_u->lnu::BR,l=tau" \
    --workers 4 \
    --samples 100000 \
    --output /tmp/unc_btotaunu.hdf5
```

As an exmple of the second way, we would like to use samples from the posterior PDF as obtain in section 2.2 in order to predict the branching ratio of the decay $\bar{B}^0 \to \pi e^-\bar{\nu}_e$, in the same kinematic range as the decay to muons described previously. Our intention translates to the following call to `eos-propagate-uncertainty`:

```
eos-propagate-uncertainty \
    --kinematics s_min  0.0 \
    --kinematics s_max 12.0 \
    --observable "B->pilnu::BR,l=e,form-factors=BCL2008" \
    --pmc-input /tmp/pmc_monolithic_btopi+ff.hdf5 0 10000 \
    --pmc-sample-directory "/data/final/" \
    --output /tmp/unc_btopilnu.hdf5
```

[1]Note that using samples from `eos-sample-mcmc` is presently not supported.

For both ways, the samples of the predictive distributions within the HDF5 files can be accessed using the `eosdata` Python module.

2.5. Plotting Random Samples

Once random samples have been obtained from either a posterior PDF (e.g. as described in section 2.2) or a predictive PDF (e.g. as described in section 2.4), 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 heatmaps of 2D (`eos-plot-2d`) PDFs. 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_`. Files containing samples from the uncertainty propagation should be prefixed with `unc_`.
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:

$$\texttt{NAMESPACE::ID@TAG}, \tag{3.1}$$

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. 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 proposed in [6] (BCL2008). 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:

$$\texttt{B->pi::f\_+(0)BCL2008},\quad \texttt{B->pi::b\_+\^{}1BCL2008},\quad \text{and}\quad \texttt{B->pi::b\_+\^{}2BCL2008}. \tag{3.2}$$

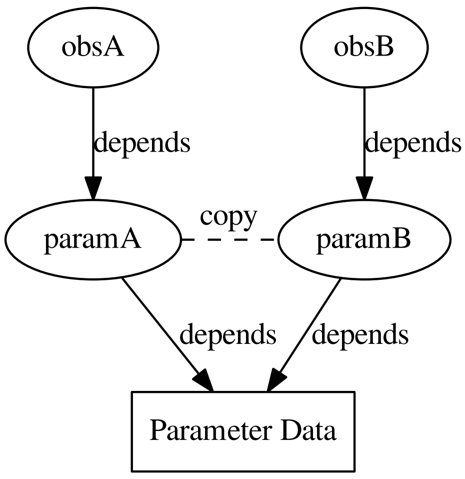


Figure 3.1.: Diagrammatic illustration that multiple observables can depend on the very same instance of `Parameters`.

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

Third, …
Likelihood and Prior

- construct likelihood and prior at run time
- abstract tree, with leaves:
  - (Multivariate) Gaussian distribution
  - LogGamma distribution (for asymmetric uncertainty intervals)
  - Amoroso (for limits)
  - Flat (prior only)

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`

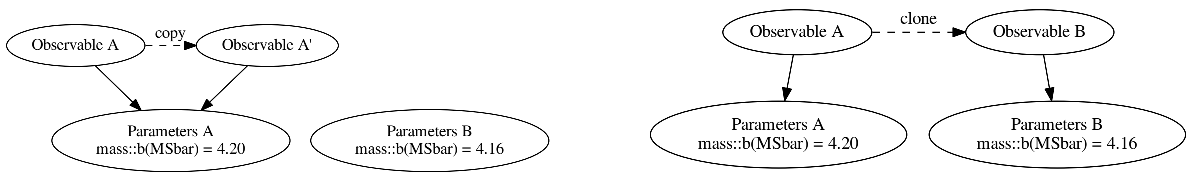
`key = value` dictionary, with string keys and string values influences how observables are evaluated

- 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



(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

3.2.4. Class `Observable`

`Observable` is an abstract base class

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

4. Extending EOS

4.1. How to add a new observable

4.2. How to add a new measurement

A. Conventions

Acronyms

LCSR Light-Cone Sum Rule. 11

PDF Probability Density Function. 9, 11, 12

PMC Population Monte Carlo. 4

RNG Random Number Generator. 9, 10

Bibliography

[1] The Python Software Foundation, *Python Language Reference, version 2.7*.

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

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

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

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

[6] 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), arXiv:0807.2722 [hep-ph].

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

[8] I. Sentitemsu Imsong, A. Khodjamirian, T. Mannel, and D. van Dyk, "Extrapolation and unitarity bounds for the $B \to \pi$ form factor", JHEP **02**, 126 (2015), arXiv:1409.7816 [hep-ph].

[9] A. Gelman and D. B. Rubin, "Inference from Iterative Simulation Using Multiple Sequences", Statist. Sci. **7**, 457 (1992).

[10] Y. Amhis et al., "Averages of b-hadron, c-hadron, and τ-lepton properties as of summer 2014", (2014), arXiv:1412.7515 [hep-ex].

[11] S. Aoki et al., "Review of lattice results concerning low-energy particle physics", Eur. Phys. J. **C74**, 2890 (2014), arXiv:1310.8555 [hep-lat].