EOS - Introduction and Overview

Danny van Dyk

New Physics at Belle II
February 24th, 2015

Bundesministerium für Bildung und Forschung

UNIVERSITÄT SIEGEN

DFG FOR 1873

What is EOS?

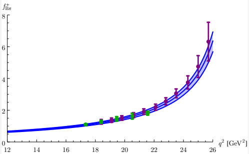
Use cases

EOS is a set of C++ libraries and programs that is used for several applications in the field of flavour physics.

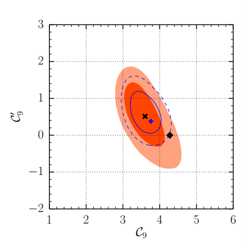
http://project.het.physik.tu-dortmund.de/eos

Use cases are:

1. evaluation of observables and related theoretical quantities, and their uncertainty estimation



2. inference of parameters from experimental or theoretical constraints



How and Who

- truely collaborative effort
- source publicly available, via tarballs and a GIT repository
  `http://project.het.physik.tu-dortmund.de/source/eos`

main authors:

- D. van Dyk (U Siegen)
- F. Beaujean (LMU Munich)
- Ch. Bobeth (TU Munich)
- S. Jahn (TU Munich)

formerly:

- Ch. Wacker

contributors:

- Th. Blake (U Warwick)
- Ch. Langenbruch (U Warwick)
- H. Miyake (U Tsukuba)
- K. Petridis (U Bristol)
- A. Shires (TU Dortmund)

Overview and Architecture

Source Tree

/ eos

- `libeos.so`: main interface to all classes

/ utils

- `libeosutils.so`: utility classes (I/O, multithreading, . . .)

/ statistics

- `libeosstatistics.so`: likelihood, Markov chains, . . .

/ b-decays

- `libeosbdecays.so`: charged-current b decays

/ rare-b-decays

- `libeosrarebdecays.so`: FCNC b decays

/ form-factors

- `libeosformfactors.so`: hadronic matrix elements

/ src

/ clients

- `eos-*`: client progams

Design: Language and Dependencies

Core library

- written in C++0x from the beginning (now C++11)
  - requires state-of-the-art GNU C++, version 4.8+
  - experimental support for LLVM clang
- built using GNU autotools, known to build on Linux and OS X
- dependencies
  - GNU Scientific Library
  - Hierarchical Data Format 5 Library

Statistics

- additional dependencies
  - Minuit2 (standalone or as part of ROOT)
  - Population Monte Carlo Library (optional, see commit 8599595)

Design: Abstraction

Everything is a parameter

- basically all quantities can be changed at run time
  - CKM Wolfenstein parameters
  - meson masses, quark masses, ...
  - hadronic matrix elements
  - life times
- allow to change role of parameter
  - estimate theory uncertainties (when treated as nuisance parameters)
  - fit from data (when treated as parameter of interest)

Plug-Ins

- most input functions can be chosen at run time
  - hadronic matrix elements (form factors) in various parametrizations
  - effective couplings (Wilson coefficients) in NP models

Design: Abstraction

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)

Design: Building Blocks

begin of a technical intermezzo

most important classes:

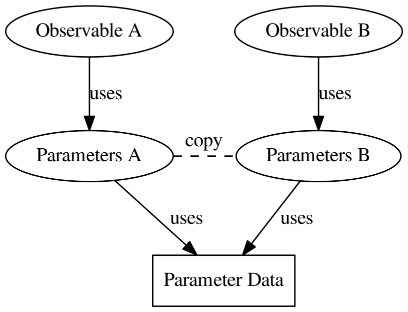
- `Parameter`, `Parameters`
- `Kinematics`
- `Options`
- `Observable`

let's go into details on each of these

Implementation: `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



Implementation: `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

- 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 KMPW '10

Implementation: `Kinematics`

`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

Implementation: `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

Implementation: Observable (concepts and interface)

`Observable` is an abstract base class

- descendants must at construction time:
  - associate with instance of Parameters
  - extract value from instance of Options
- construction via factory method:
  create an observable at runtime using its name, a set of parameters, a set of kinematic variables, and a set of options:

  ```
  Observable::make("B->pilnu::BR", p, k, o)
  ```
- changes to `Parameters` transparently affect associated observables
- changes to `Options` do not affect the observable after construction

Implementation: Observable (concepts and interface)

- observables can be
  - evaluated:

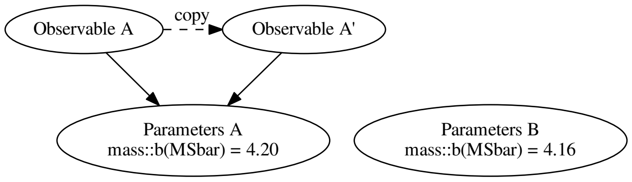
    runs the necessary computations for the present values of the parameters
  - copied:

    copy-ctor does not create an independent copy, the copy uses the same parameters, with the same options
  - cloned:

    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
  - easily allows to parallelize algorithms

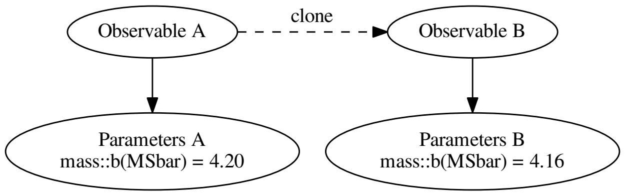
Implementation: Observable (concepts and interface)

Copying



Implementation: Observable (concepts and interface)

Cloning



Implementation: Observable (example)

example of an observable:

- class `BToPiLeptonNeutrino`
  - inherits from `PrivateImplementationPattern`
    $\Rightarrow$ Copy-CTOR does not produce independent copy
- method
  `integrated_branching_ratio(const double & s_min, const double & s_max)`
  - two kinematic variables: `s_min`, `s_max`
    correspond to integration range for $\ell\bar{\nu}$ mass square
- `Observable::make(...)` associates name **`B->pilnu::BR`** with an instance of `BToPiLeptonNeutrino` and its method `integrated_branching_ratio`
  - associates kinematic variable `s_min` with first argument, `s_max` with second argument

end of the intermezzo

Adding Observables

easy to add new observables, with efficient implementation:

- core library provides commonly-used functions
  - α_s running ($\overline{\text{MS}}$ scheme)
  - quark-mass running and scheme conversion ($\overline{\text{MS}}$, kinetic and pole schemes)
  - Wilson coefficients of $b \to s$ EFT in the SM
  - Pion light-cone distriubution amplitudes (twists 2 through 4)
- memoisation of expensive function calls
  - created table at run time, lookup of result for known arguments
  - easy-peasy:

```
expensive_function(arg1, arg2, ..., argN);
```

$\Downarrow$

```
memoise(expensive_function, arg1, arg2, ..., argN);
```

Parameter Inference

- parameter inference is EOS' 2nd use case
- setup for Bayesian anaylses
  - mode-finding accomplished using Minuit2
    - planned: allow for configurable mode-finding libraries
  - integration of the posterior is the leading numerical problem
  - using black-box algorithm that works for large parameter spaces
    - works up to approx. 40 - 50 parameters
    - adaptive importance sampling
    - uses Markov Chains and Population Monte Carlo

Usage

mainly three clients = programms that use EOS library

- evaluation via eos-evaluate
  - takes a list of observables and their kinematics from command line
  - outputs table of evaluation to STDOUT
  - naive error estimation available, assumes Gaussian errors
- parameter inference and sampling via eos-scan-mc
  - Bayesian inference, constructs prior and likelihood from command line
  - accesses EOS' library of expt. constraints and theory inputs
  - outputs posterior samples to HDF5 file
- eos-propagate-uncertainty
  - draws samples from a predictive distribution
  - takes posterior samples from HDF5 file or prior samples from command line
    - combination possible!
  - outputs samples to HDF5 file

Usage

- carry out analyses via command line
- internally: use set of BASH/Python scripts
  - runs EOS analyses on laptops, workstations or clusters
  - works on at least two different clustering systems
  - happy to share them, please approach us

Walkthrough of a Recent Analysis

$B \to \pi$ Form Factor and $|V_{ub}|$ from $\bar{B}^0 \to \pi^+\mu^-\bar{\nu}$

let's look at a small-scale study that uses EOS

- walkthrough of recent study Imsong/Khodjamirian/Mannel/DvD 1409.7816
- $B \to \pi$ form factor $f_+^{B\pi}(q^2)$ from Light-Cone Sum Rules (LCSRs)
  - first LCSR result that provides correlations of parametric uncertainties
- one application: determination of $|V_{ub}|$ from BaBar and Belle measurements of $\bar{B}^0 \to \pi^+\mu^-\bar{\nu}$

(for large-scale study, see Christoph Bobeth's talk)

Step 1: Implementation

- implement $f_+^{B\pi}$ from Light-Cone Sum Rules (LCSRs)
  Duplancic/Khodjamirian/Mannel/Melic/Offen 0801.1796
- add `AnalyticBToPiFormFactorsDKMMO2008`
  - introduce relevant input parameters to `Parameters`
  - implement $f_+(q^2)$, $f_+'(q^2)$ and $f_+''(q^2)$ for predictions
  - implement 2 ancillary observables for theory constraints
  - implement target observable $\mathcal{B}(\bar{B}^0 \to \pi^+\mu^-\bar{\nu}_\mu)$

Step 1: Implementation

- implement $f_+^{B\pi}$ from Light-Cone Sum Rules (LCSRs)
  Duplancic/Khodjamirian/Mannel/Melic/Offen 0801.1796
- add `AnalyticBToPiFormFactorsDKMMO2008`
  - introduce relevant input parameters to `Parameters`
  - implement $f_+(q^2)$, $f'_+(q^2)$ and $f''_+(q^2)$ for predictions
  - implement 2 ancillary observables for theory constraints
  - implement target observable $\mathcal{B}(\bar{B}^0 \to \pi^+\mu^-\bar{\nu}_\mu)$

/ eos

modified files

- M observable.cc
- / b-decays
  - M `Makefile.am`
  - + `b-to-pi-l-nu.cc`, `b-to-pi-l-nu.hh`
- / form-factors
  - M `Makefile.am`, `mesonic.hh`, `mesonic.cc`
  - + `analytic-b-to-pi.cc`, `analytic-b-to-pi.hh`
- / utils
  - M `parameters.cc`

Step 2: $B \to \pi$ Form Factor

- construct PDF for the input parameters
  - 16 dim. parameter space
  - uncorrelated priors individual parameters: m_b, f_π, ...
  - includes two theory constraints
- mode-finding and integration using eos-scan-mc
  - 16 Markov chains, run independently
  - combine chains, and initialize PMC with 4 clusters
  - PMC converged after 2 update steps
  - draw $5 \cdot 10^4$ samples from the posterior
- eos-propagate-uncertainty: compute posterior-predictive distribution for $B \to \pi$ form factor and its derivative
  - distribution of f_+ and derivatives is Gaussian to very good approximation
  - estimate covariance from samples

Step 3: $|V_{ub}|$ from $\bar{B}^0 \to \pi^+ \mu^- \bar{\nu}$

- add constraints to `libeos.so`
  - add theory constraint `B->pi::f_+@IKMvD2014` based on previous results $\Rightarrow$ can be reused for future projects ($B \to \pi \ell^+ \ell^-$!)
  - implement experimental constraints `B->pilnu::BR@*` based on various BaBar and Belle measurements
- fit $|V_{ub}|$ and form factor parameters to exp.&th. constraints, using `eos-scan-mc`
  - 16 Markov chains explore parameter space
  - combine chains and initialize PMC w/ 4 clusters
  - PMC converged after 3 update steps
  - draw 10^5 samples from the posterior, effective sample size: 94%

Step 3: $|V_{ub}|$ from $\bar{B}^0 \to \pi^+\mu^-\bar{\nu}$

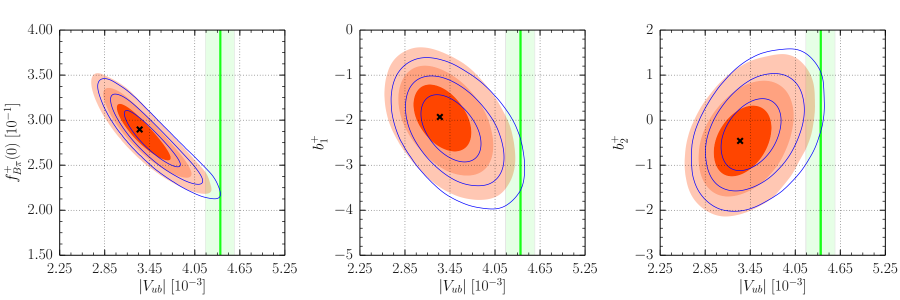
- add constraints to `libeos.so`
  - add theory constraint `B->pi::f_+@IKMvD2014` based on previous results
    $\Rightarrow$ can be reused for future projects ($B \to \pi\ell^+\ell^-$!)
  - implement experimental constraints `B->pilnu::BR@*` based on various BaBar and Belle measurements
- fit $|V_{ub}|$ and form factor parameters to exp.&th. constraints, using `eos-scan-mc`
  - 16 Markov chains explore parameter space
  - combine chains and initialize PMC w/ 4 clusters
  - PMC converged after 3 update steps
  - draw 10^5 samples from the posterior, effective sample size: 94%

/ eos
  M constraint.cc

modified files

Step 3: $|V_{ub}|$ from $\bar{B}^0 \to \pi^+\mu^-\bar{\nu}$

produce pretty plots



First determination of $|V_{ub}|$ from Light-Cone Sum Rules with Bayesian treatment of parametric theory uncertainties

blue contours	68%, 95%, 99% prob. contours for 2010 data (BaBar+Belle)
red areas	68%, 95%, 99% prob. contours for 2013 data (BaBar+Belle)
green vertical line/area	central value/68% CL interval for $B \to X_u\ell\bar{\nu}$ (GGOU/HFAG)

Conclusion and Outlook

Conclusion

- EOS is a HEP flavour program for observable evaluation and parameter inference
  - adding observables is rather eady
  - reduces code replication by sharing common code (RGE running, . . .)
- already contains theory codes for many interesting observables
  - $b \to s\ell^+\ell^-$, $b \to s\gamma$: excl. and incl. decays, see Ch. Bobeth's talk
  - $b \to u\ell\bar{\nu}$ (exclusive only)
- powerful black-box algorithm for mode-finding and posterior integration
  - allows for treatment of th. uncertainties via nuisance parameters
  - for algorithm see F. Beaujean's dissertation (link in backups)

Belle II and EOS

- mutually benefitial exchange with members of LHCb
  - discussions with Belle II members would be very welcome
  - tell us if you want use EOS
  - tell us about your applications
  - patches/contributions always welcome!
- prospects
  - Feature: EOS as an event Monte Carlo generator in NP models?
    importance sampling already in place in the library; basically needs only a new client program
  - Feature: Python interface
  - Physics: $B \to X_c \ell \bar{\nu}$ observables w/ comprehensive theory uncertainty?
  - Physics: $b \to \{c, u\} \ell \bar{\nu}$ in EFT?
- what would you consider helpful or important?

Backup Slides

Sketch of the Black-Box Algorithm

Algorithm as described in Beaujean 2012[1]
basic idea:

1. let adaptive markov chains explore the paramater space
   - usual problem: chains do not mix $\Rightarrow$ chains may be biased towards individual modes
   - solution: chop chains up into patches, let hierarchical clustering sort patches into clusters
   - extract mode and covariance for each cluster
2. create mixture density based on modes and covariances for each cluster
3. use population monte carlo to find mixture density that approximates the target
   - draws samples from approximative results, compares with target
   - each update step decreases Kullback-Leibler divergence between approximation and target
   - final step: draw weighted variates from approximation

[1] http://d-nb.info/1031075380