EOS Tutorial

EOS Collaboration

Beaujean / Bobeth / van Dyk

B2TIP meeting on theory codes

LAL

Outline

- Introductory remarks
- General fitting strategy in **EOS**
- **EOS** client "**eos-scan-mc**" for fits
- Fits with Markov Chains (MCMC)
- Fits with Population MC (PMC)
- Other use cases — not covered

Introductory Remarks

EOS — Websites and Source Code

EOS homepage — **http://project.het.physik.tu-dortmund.de/eos/**

- list of talks and papers

EOS @ GitHub — **https://github.com/eos**

- git-repo with source code
- You are welcome to participate !

EOS — Languages and Dependences

Core library + client programs

- 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
- dependences
  - GNU scientific Library (GSL)
  - Hierarchical Data Format 5 Library (HDF5)

Statistics

- Minuit2 (standalone or as part of ROOT)
- Population Monte Carlo Library **`pmclib`** (optional, see commit 8599595) required to perform fits with PMC [Kilbinger et al. **`http://www2.iap.fr/users/kilbinge/CosmoPMC`**]
- in developement: external fitter **`pypmc`** + interface to **EOS**

EOS — Manual

Now available: very brief first version of manual @ GitHub

```
> make manual
```

or

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

- installation instructions
- usage of **EOS**-clients
  1) **eos-evaluate**
  2) **eos-scan-mc** ← *fits in this tutorial*
- description of core desing of **EOS** library to facilitate implementation of own observables

EOS — Source Tree

https://github.com/eos

/eos main interface to all classes — **libeos.so**

/utils/... — **libeosutils.so**

/parameters.cc

utility classes (I/O, multithreading, . . .)

/statistics/... — **libeosstatistics.so**

likelihood, Markov chains, . . .

/b-decays/... — **libeosbdecays.so**

charged-current b decays: $b \to (u,\ c) + \ell\bar{\nu}_\ell$

/rare-b-decays/... — **libeosrarebdecays.so**

FCNC b decays: $b \to s + (\gamma,\ \bar{\ell}\ell)$

/form-factors/... — **libeosformfactors.so**

form factors + other hadronic matrix elements for b decays

/observable.cc observable factory: bind a “name” to a function

/constraint.cc bind measurments + correlation to observables, with specific kinematics + options

/references.txt the references appearing in comments to the code

/src/clients/... client programs

/eos-evaluate

/eos-list-parameters

/eos-scan-mc ← *client used for fits*

...

[see PNNL-tutorial Ryosuke Itoh]

/src/scripts/... simple plot scripts ← *visualise results*

General fitting strategy with EOS

EOS statistics library is "Bayesian"

- **data** D
- a **model** M with
  - **parameters of interest** $\vec{\theta}$
  - **nuisance parameters** $\vec{\nu}$
  - !!! conceptually there is no difference between $\vec{\theta}$ and $\vec{\nu}$
- **prior** pdf's (probability distr. functions) for $\vec{\theta}$ and $\vec{\nu}$ given model M $\quad P(\vec{\theta}, \vec{\nu} \,|\, M)$
- **likelihood** of D given model M with parameters $(\vec{\theta}, \vec{\nu})$ $\quad P(D \,|\, M, \vec{\theta}, \vec{\nu})$
- normalization factor (**evidence**) $\quad Z = \int \mathrm{d}\vec{\nu}\, \mathrm{d}\vec{\theta}\; P(D \,|\, M, \vec{\theta}, \vec{\nu})\; P(\vec{\theta}, \vec{\nu} \,|\, M)$

  $\Rightarrow$ allows model comparison between several models: $M_1, M_2, \ldots$

Bayes Theorem infer **posterior** pdf of $\vec{\theta}$ – marginalising over $\vec{\nu}$ – given data D and model M

$$P(\vec{\theta} \,|\, M, D) = \frac{\int \mathrm{d}\vec{\nu}\; P(D \,|\, M, \vec{\theta}, \vec{\nu})\; P(\vec{\theta}, \vec{\nu} \,|\, M)}{Z}$$

In **EOS** this problem is *solved numerically with Monte Carlo methods*

- random walk: Markov Chains with adaptive Metropolis-Hastings (MCMC)
- importance sampling: Population Monte Carlo (PMC)

EOS Fits: in 3 steps MCMC + HC + PMC

described in [Beaujean PhD thesis, Beaujean/Caldwell 1304.7808]

used in [Beaujean/Bobeth/van Dyk/Wacker 1205.1838, Beaujean/Bobeth/van Dyk 1310.2478v3, Imsong/Khodjamirian/Mannel/van Dyk 1409.7816, Feldmann/Müller/van Dyk 1503.09063]

A) Markov Chain pre-run (MCMC)

Multiple MC's run (in parallel) using Metropolis-Hastings to explore parameter space

- chains are started from random point drawn from prior
- number of chains must be optimised by user
- parallelization is limited to parallel run of chains
  $\Rightarrow$ a chain itself can not be parallelized due to serial nature of Markov process

Advantage: allows very efficient localisation and exploration of local modes

Problem: in multi-modal target density MC's usually trapped in local modes

$\Rightarrow$ MC's are not sufficiently mixed to be combined to single MC

$\Rightarrow$ criteria for mixing: Gelman-Rubin R-value

Disadvantage: no straight forward calculation of "evidence" for model comparison

EOS Fits: in 3 steps MCMC + HC + PMC

described in [Beaujean PhD thesis, Beaujean/Caldwell 1304.7808]

used in [Beaujean/Bobeth/van Dyk/Wacker 1205.1838, Beaujean/Bobeth/van Dyk 1310.2478v3, Imsong/Khodjamirian/Mannel/van Dyk 1409.7816, Feldmann/Müller/van Dyk 1503.09063]

B) Hierarchical clustering (HC)

Transform Markov Chain's into mixture density of multi-variate gaussian functions as initialisation of importance sampling PMC

- group MC chains using R-value (should correspond to local modes)
- split chains into sub-chains (patch) and generate components from their samples (component = multi-variate gaussian)
- use hierarchical clustering [Goldberger/Roweis Adv.Neur.Info.Proc.Syst. 17 (2004) 505] to combine components that are "redundant" based on Kullback-Leibler divergence

Advantage: allows to eliminate redundant components and reduce their number

Disadvantage: user needs to determine the final number of components (our rule of thumb: should be at least as large as dimension of parameter space)

$\Rightarrow$ "Variational Bayes" automatically determines number of relevant components
$\rightarrow$ implemented in **`pypmc`**

EOS Fits: in 3 steps MCMC + HC + PMC

described in [Beaujean PhD thesis, Beaujean/Caldwell 1304.7808]

used in [Beaujean/Bobeth/van Dyk/Wacker 1205.1838, Beaujean/Bobeth/van Dyk 1310.2478v3, Imsong/Khodjamirian/Mannel/van Dyk 1409.7816, Feldmann/Müller/van Dyk 1503.09063]

C) Importance sampling via Population Monte Carlo (PMC)

- initialised with mixture density determined in MCMC + HC
  - ⇒ all components have equal weight
    (balance effect of unequal number of chains in local modes)
  - ⇒ can replace (all) gaussian components by student-T
    (with optional choice of fixed degrees of freedom → heavier tails)

- PMC algorithm proceeds iteratively
  1) draw samples from current mixture density
     (number of samples user choice, min. number of samples per component required)
  2) calculate new weights of components based on PMC algorithm

     [Cappé/Douc/Guillin/Martin/Robert 0710.4242]

     [Wraith/Kilbinger/Benabed/Cappé/Cardoso/Fort/Prunet/Robert 0903.0837]

  3) check convergence of “perplexity” and “effective sample size”

- draw larger set of samples in final step

EOS — Parallelization

- threading on single multi-core machine possible
- parallelization of MCMC trivial (→ hierarchical clustering merges chains later on)
- parallelization of PMC possible via threads on single computer (**`"PMC-monolithic"`**)

  BUT otherwise highly dependent on queuing system of available cluster

  ⇒ achieved by multiple runs of **`eos-scan-mc`**

  ⇒ python script used for steering of PMC for

  1) sampling step
  2) update step of mixture density
  3) convergence check

Please contact us for assistance

EOS client

“eos-scan-mc”

EOS-client "`eos-scan-mc`"

- located in `.../src/clients`
- the whole fit is **configured** by calling `eos-scan-mc` **via command line options**
  $\Rightarrow$ best to look at source code for all available options
- we use shell-scripts to set up and steer fits

There are options for

1) physics analysis: models, parameter priors, experimental measurements
2) seeds / input / output / debug
3) MCMC: prerun, main run, number of chains, number of samples
4) PMC: initialisation via HC, PMC update steps, PMC final samples
5) goodness-of-fit (GOF)
6) uncertainty-propagation
7) ...

The command line (CMDLINE) is a stream of instructions (stateful):

Values of options and kinematics are valid for upcoming constraints and observables until they are set to different values

CMDLINE options: Fit parameters . . .

Declare fit parameters . . .

- **EOS** knows parameters by NAME, which are listed by client

  `.../src/clients/eos-list-parameters`

  together with default values

- you tell **EOS** to fit a **parameter of interest** by `--scan` NAME

  or a **nuisance parameter** `--nuisance` NAME

!!! `--nuisance` works identical to `--scan`, except special flag in HDF5 output for potentially separate handling in analysis

- all parameters, which are NOT fitted take fixed values – usually default

  ⇒ to override default values in `.../eos/utils/parameters.cc` use

  `--fix` NAME VALUE

CMDLINE options: . . . and priors

. . . followed by their prior only 1-dim priors via CMDLINE

- **flat prior** in interval [MIN, MAX] `--prior flat` MIN MAX

  or MIN MAX `--prior flat`

Examples

```
--scan "CKM::rhobar" --prior flat 0.083 0.181
```

```
--scan "CKM::rhobar" 0.083 0.181 --prior flat
```

CMDLINE options: . . . and priors

. . . followed by their prior only 1-dim priors via CMDLINE

- **gaussian prior** with
  1) central value CEN,
  2) left and right std dev's LSIG = CEN - MIN and RSIG = MAX - CEN
  3) support of gaussian prior does NOT extend from $[-\infty, +\infty]$, but is restricted to NSIG $\times$ (LSIG, RSIG) with NSIG < 10
  4) optional to provide support as interval [ABSMIN, ABSMAX]

  $\Rightarrow$ **EOS** chooses smallest range between 3) and 4)

  !!! asymmetric (LSIG$\neq$RSIG) gaussians are continuous

`--scan` NAME NSIG `--prior gaussian` MIN CEN MAX

`--scan` NAME ABSMIN ABSMAX NSIG `--prior gaussian` MIN CEN MAX

Examples

```
--scan "CKM::rhobar" 3 --prior gaussian 0.083 0.132 0.181
```

$\Rightarrow$ here support of prior would start at $0.132 - 3 \times (0.132 - 0.083) = -0.015$

```
--scan "CKM::rhobar" 0.0 1.0 3 --prior gaussian 0.083 0.132 0.181
```

$\Rightarrow$ can be avoided with ABSMIN = 0.0; ABSMAX = 1.0 is not effective since NSIG = 3

CMDLINE options: . . . and priors

. . . followed by their prior only 1-dim priors via CMDLINE

- a **log-Gamma prior** $\Rightarrow$ kind of asymmetric gaussian with different tails

`--scan` NAME [ABSMIN ABSMAX] NSIG `--prior log-gamma` MIN CEN MAX

Examples

```
--scan "CKM::rhobar" 0.0 1.0 3 --prior log-gamma 0.083 0.132 0.181
```

CMDLINE options: Constraints

Declare Data = Measurement or Theoretical constraints

- the NAME's of all exp. measurements and theo. constraints in `.../eos/constraint.cc`
- the names of all observables OBS in `.../eos/observable.cc`
- pdf's for constraints currently implemented in **EOS**
  1) multi-variate gaussian pdf's
  2) 1-dim Amoroso pdf's ⇐ to model upper bounds
- you can **select constraints** for analysis with `--constraint` NAME
- sometimes might difficult to implement a prior for an observable or theory-quantity OBS
  ⇒ create a gaussian pdf with cen val CEN & left/right std. dev's (CEN - MIN, MAX - CEN)
  as (pseudo)-data via

  `--observable` OBS MIN CEN MAX

  `--observable-prior` OBS MIN CEN MAX

  !!! `--observable` counts as degree of freedom in GOF, `--observable-prior` does not

If OBS has kinematics KIN1, KIN2, ... with values V1, V2, ..., specify beforehand

`--kinematics` KIN1 V1 `--kinematics` KIN2 V2 ... `--observable` OBS MIN CEN MAX

Examples

```
--constraint "B^0_s->mu^+mu^-::BR@CMS-LHCb-2014"
--kinematics s 0.0 -observable "B->K^*::V(s)/A_1(s)" 0.93 1.33 1.73
```

CMDLINE options: Models etc.

Global options

- specify a model MODEL via `--global-option model` MODEL

  currently MODEL =

`SM`	Standard Model (SM
`CKMScan`	CKM parameters $\lvert V_{ij}\rvert$ and arg V_{ij} in SM
`WilsonScan`	fits of $\Delta B = 1$ Wilson coeff's
`ConstrainedWilsonScan`	fits of $\Delta B = 1$ Wilson coeff's with add. constraints

- specify parameterisation PARAM of complex Wilson coefficients via `--global-option scan-mode` PARAM

  with PARAM = `cartesian` (default) or `polar`

- specify q^2-parameterisation PARAM of form factors `--global-option form-factors` PARAM

  currently PARAM =

`KMPW2010`	[Khodjamirian et al. 1006.4945]
`BSZ2015`	[Bharucha et al. 1503.05534]
`...`	check source code (we stil need to document this; if in doubt, ask Danny)

CMDLINE options: seeds / output / input / etc

Seed value for random number generator (RNG)

- for reproducibility — not only while testing — can initialise RNG with a seed-VALUE

`--seed` VALUE

Output-file name

- output files are in HDF5 format
- contain meta information about the analysis (priors, constraints, . . .)
- for MCMC runs: samples of Markov chains (pre- and main-runs)
- for PMC runs: samples + components and their weights
- use `--output` FILENAME

Input-file name

- MCMC prerun output needed for initialisation of PMC

`--pmc-initialize-from-file` FILENAME

Additional output

- for additional detailed output use `--debug`

Example: Fit C_{10} with $Br(B_s \to \mu^+\mu^-)$

Fit (real-valued) $b \to s\bar{\mu}\mu$ Wilson coefficient $C_{10}(\mu_b = 4.2\,\text{GeV})$ SM prediction $C_{10}^{\text{SM}} \simeq -4.2$

- from LHCb + CMS 2014 measurement of

$$Br(B_s \to \bar{\mu}\mu) \propto f_{B_s}^2 \times |V_{tb}V_{ts}^*|^2 \times |C_{10}|^2$$

  $\Rightarrow$ due to quadratic dependence there are two solutions for C_{10}

- additional nuisance parameters:

  1) CKM Wolfenstein parameters λ, A, $\bar{\rho}$, $\bar{\eta}$

  2) B_s-decay constant f_{B_s}

!!! Currently you are probably in directory **`.../src/clients`**

$\Rightarrow$ move to a dedicated directory, and make sure that installation directory of the **EOS** clients and scripts is accessible via PATH
to keep results separated from **EOS** source code

Example: C_{10} in $B_s \to \mu^+\mu^-$ — CMDLINE input

!!! Be aware that CMDLINE parsing not foolproof ⇒ best to declare analysis in following order

1) general instructions
2) scan and nuisance parameters with corresponding global options
3) constraints with corresponding global options
4) constraints on observables with corresponding global options

```
> eos-scan-mc \
    --seed 1234 \
    --global-option model WilsonScan \
    --global-option scan-mode cartesian \
    --scan Re{c10} -7.0 -1.0 --prior flat \
    --nuisance CKM::lambda         3 --prior gaussian 0.2247 0.2253 0.2259 \
    --nuisance CKM::A              3 --prior gaussian 0.787 0.807 0.827 \
    --nuisance CKM::rhobar 0.0 1.0 3 --prior gaussian 0.073 0.128 0.183 \
    --nuisance CKM::etabar         3 --prior gaussian 0.315 0.375 0.435 \
    --nuisance decay-constant::B_s 3 --prior gaussian 0.2232 0.2277 0.2322 \
    --constraint "B^0_s->mu^+mu^-::BR@CMS-LHCb-2014" \
    --output mcmc_c10.hdf5
```

- **`eos-scan-mc`** uses some default settings (which are not shown here)
- a seed value is used for reproducibility
- there is some output on the screen during the run
- an output file **`"mcmc_c10.hdf5"`** is generated

Example: C_{10} in $B_s \to \mu^+\mu^-$ — MCMC runtime output

The current analysis configuration

```
# Scan generated by eos-scan-mc
# Scan parameters (1):
#   Parameter: Re{c10}, prior type: flat, range: [-7,-1]
# Nuisance parameters (5):
#   Parameter: CKM::A, prior type: Gaussian, range: [0.747,0.867], x = 0.807 +- 0.02
#   Parameter: CKM::lambda, prior type: Gaussian, range: [0.2234,0.2273], x = 0.22535 +- 0.00065
#   Parameter: CKM::rhobar, prior type: Gaussian, range: [0,0.293], x = 0.128 +- 0.055
#   Parameter: CKM::etabar, prior type: Gaussian, range: [0.195,0.555], x = 0.375 +- 0.06
#   Parameter: decay-constant::B_s, prior type: Gaussian, range: [0.2126,0.2426], x = 0.2276 +- 0.005
# Constraints (1):
#  B^0_s->mu^+mu^-::BR@CMS-LHCb-2014: B_q->ll::BR@Untagged[] with options: l=mu,model=WilsonScan,q=s,scan
      -mode=cartesian, Amoroso limit: mode at B_q->ll::BR@Untagged = 2.8e-09 (a = 0, theta = 1.5424e-10,
      alpha = 19.402, beta = 1.0048)
...
```

- can verify correct settings for scan and nuisance parameters
- details on constraints

Example: C_{10} in $B_s \to \mu^+\mu^-$ — MCMC runtime output

Information on MCMC pre-run

```
...
...: [INFO markov_chain_sampler.initialize] Determining initial proposal covariance assuming flat priors
...: [INFO markov_chain_sampler.initialize] Using proposal_functions::MultivariateGaussian
...: [INFO markov_chain_sampler.prerun_start] Commencing the pre-run with 400, 100000, 400 (min, max,
      update) iterations.
...: [INFO prop::Multivariate.adapt] Change scale from 0.9440666667 to 0.6293777778
...: [INFO prop::Multivariate.adapt] Change scale from 0.9440666667 to 0.6293777778
...: [INFO prop::Multivariate.adapt] Change scale from 0.9440666667 to 0.6293777778
...: [INFO prop::Multivariate.adapt] Change scale from 0.9440666667 to 0.6293777778
...: [INFO markov_chain_sampler.parameter_rvalue_too_large] R-value of parameter 'Re{c10}' is too large:
      1.198462894 > 1.1
...: [INFO markov_chain_sampler.parameter_rvalue_too_large] R-value of parameter 'CKM::A' is too large:
      1.243953594 > 1.1
...: [INFO markov_chain_sampler.parameter_rvalue_too_large] R-value of parameter 'CKM::lambda' is too
      large: 1.132905817 > 1.1
...: [INFO markov_chain_sampler.parameter_rvalue_too_large] R-value of parameter 'CKM::rhobar' is too
      large: 1.218825872 > 1.1
...: [INFO markov_chain_sampler.parameter_rvalue_too_large] R-value of parameter 'CKM::etabar' is too
      large: 1.129397518 > 1.1
...: [INFO markov_chain_sampler.prerun_progress] Pre-run has completed 400 iterations
...: [INFO markov_chain_sampler.efficiencies] All efficiencies OK
...: [INFO markov_chain_sampler.convergence] All R-values OK
...: [INFO markov_chain_sampler.convergence] Convergence achieved
...: [INFO markov_chain_sampler.prerun_progress] Pre-run has completed 800 iterations
...: [INFO markov_chain_sampler.prerun_converged] Pre-run has converged after 800 iterations
```

- starts by default a MCMC with 4 chains with pre-run to optimise the proposal function
- in the example the pre-run finds optimal efficiencies after 400 iterations
- however, the 4 chains mix sufficiently "only" after 800 iterations, i.e. all R-values < 1.1

Example: C_{10} in $B_s \to \mu^+\mu^-$ — MCMC runtime output

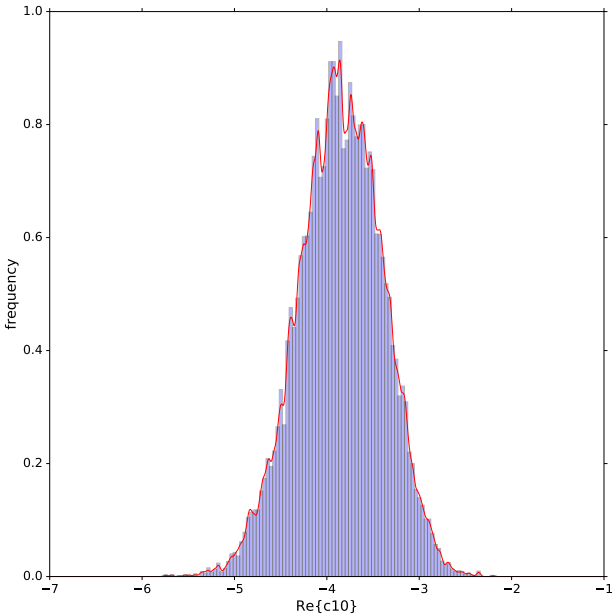
and MCMC main-run

```
...: [INFO markov_chain_sampler.mainrun_start] Commencing the main-run
...: [INFO markov_chain_sampler.mainrun_progress] Main-run has completed 1000 iterations
...: [INFO markov_chain_sampler.convergence] Checking R-values for the last chunk of size 1000
...: [INFO markov_chain_sampler.main_run] All R-values OK
...
...: [INFO markov_chain_sampler.mainrun_progress] Main-run has completed 10000 iterations
...: [INFO markov_chain_sampler.convergence] Checking R-values for the last chunk of size 1000
...: [INFO markov_chain_sampler.main_run] All R-values OK
...: [INFO markov_chain_sampler.mainrun_end] Finished the main-run
```

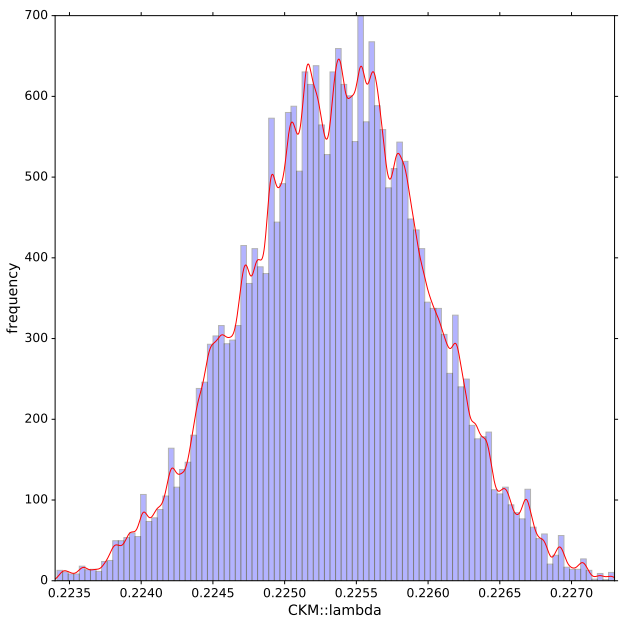
- main-run proceeds in 10 (=chunks) steps of 1.000 iterations (=chunks-size) until 10.000 (default values)
- R-values are checked after each chunk
  $\Rightarrow$ OK means that the 4 chains sample “same part of parameter space”
- exits after main-run with generation of output file **`"mcmc_c10.hdf5"`**

Example: C_{10} in $B_s \to \mu^+\mu^-$ — Output "mcmc_c10.hdf5"

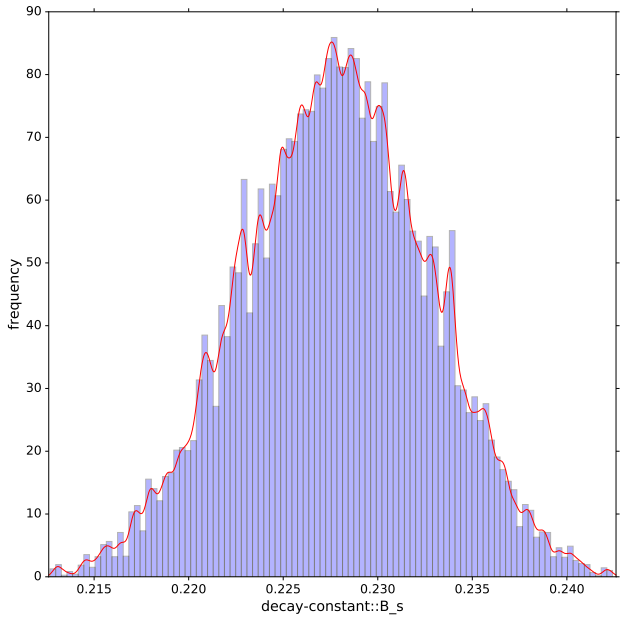
- the HDF5 file contains all meta info of analysis and samples of MCMC pre- and main-runs
  $\Rightarrow$ might browse with some HDF5 viewer through the file
- generate histogram with simple **python**-script located in **/src/scripts**
  > python ../scripts/eos-plot-1d HDF5IN IDX PDFOUT
  parameter-index IDX corresponds to position in Cmd-line: 0 = **Re{c10}**, 1 = **CKM::A**, ...
  > python ../scripts/eos-plot-1d mcmc_c10.hdf5 0 c10_0.pdf



C_{10}



CKM::λ



f_{B_s}

Options for MCMC

MCMC options

- set the NUMBER of Markov chains
  `--chains` NUMBER or **`--prerun-chains-per-partition`** NUMBER

Pre-run

- min NUMBER of steps of pre-run **`--prerun-min`** NUMBER
- max NUMBER of steps of pre-run **`--prerun-max`** NUMBER
- force every NUMBER of steps an adaption of proposal function **`--prerun-update`** NUMBER
- if adaption too slow, might change this VALUE **`--scale-reduction`** VALUE
- when a MCMC main-run not needed (perhaps main-run with PMC) **`--prerun-only`**
- store pre-run samples to output file **`--store-prerun`** [**`0`** | **`1`**]

Main-run

total number of steps in Markov chain = chunks × chunk-size

- set NUMBER of chunks **`--chunks`** NUMBER
- set NUMBER of steps per chunk **`--chunk-size`** NUMBER

MCMC — multimodal target density

1) MCMC is good to **explore your problem**: uni-modal or several distinct solutions?
2) **diagnose with pre-run**: did chains converge, i.e. R < 1.1 for all chains?
3) **in case of uni-modal** problems can proceed with MCMC main-run

Use **previous example $B_s \to \bar{\mu}\mu$** with small change of prior and output file

```
> eos-scan-mc ... --scan Re{c10} -6.0 +6.0 --prior flat ... --output mcmc_c10-B.hdf5
```

```
...
: [INFO markov_chain_sampler.efficiencies] All efficiencies OK
: [INFO markov_chain_sampler.parameter_rvalue_too_large] R-value of parameter 'Re{c10}' is too large:
      15.94862616 > 1.1
: [INFO markov_chain_sampler.prerun_progress] Pre-run has completed 100000 iterations
: [WARNING markov_chain_sampler.no_convergence] Pre-run did NOT converge!
...
```

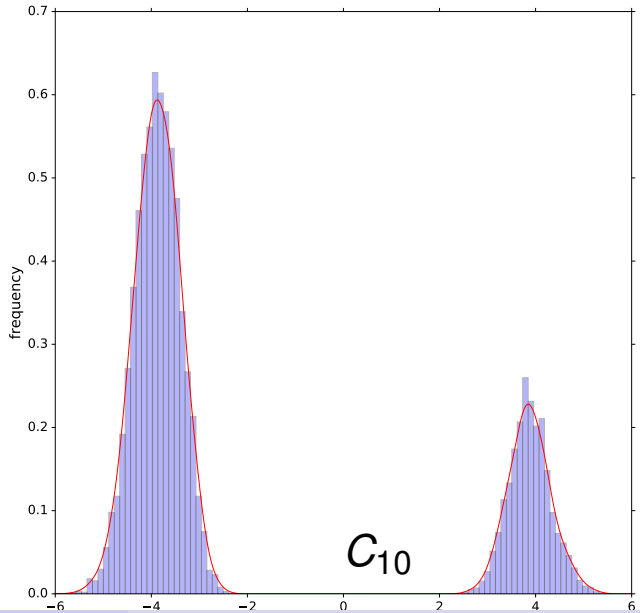
- you will find that **pre-run did not converge** and R-value too large

⇒ Markov chains (MC) did not explore entire parameter space because of **two distinct solutions**

- 3 MC's in left mode

  only 1 MC in right mode

  ⇒ wrongly weighted ⇒ **need PMC**



Options for PMC

CMDLINE options: PMC general options

- to use pmc `--use-pmc`
- use MCMC pre-run from FILENAME to initialise PMC
  `--pmc-initialize-from-file` FILENAME

Other options are available for massive parallelisation via some external steering script in order to resume pmc

- for update step and convergence check
- sampling steps
- final sampling step
- choose function-type for components of density mixture (gaussian, student-T)

CMDLINE options: Initialisation of PMC = Hierarchical clustering

- at which RVALUE (> 1) Markov Chains (MC) from prerun belong to the same group

  (this is clustering algorithm via *R* value)

  `--pmc-group-by-r-value` RVALUE

- NUMBER of target components that represent a group in the PMC run

  `--hc-target-ncomponents` NUMBER

- chop the MC's from MCMC prerun into patches with this NUMBER of samples

  `--hc-patch-length` NUMBER

- skip the first FRACTION ($\in [0, 1]$) of the MCMC result (drop the burn in phase of pre-run)

  `--hc-skip-initial` FRACTION

CMDLINE options: PMC options

- the NUMBER of samples drawn per component during PMC update steps

  `--pmc-samples-per-component` NUMBER

- the NUMBER of final samples to be drawn, once PMC converged to optimum

  `--pmc-final-samples` NUMBER

- for BOOL = 0 the samples of components with zero weight (“dead components”) get redistributed to maintain total number of samples constant
  for BOOL = 1 number of samples per components constant

  `--pmc-adjust-sample-size` BOOL

- effective sample size (ESS) is ignored as convergence criteria for BOOL = 1

  `--pmc-ignore-ess` BOOL

- fixes the maximal standard deviation MAXSTD over the last number of PMC STEPS for perplexity (ESS optional) as convergence criterium for the PMC run

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

Example: C_{10} in $B_s \to \mu^+\mu^-$ — CMDLINE for PMC fit

Launch PMC fit with following CMDLINE:

```
> eos-scan-mc \

  --seed 1234 \                                                                  general
  --debug \
  --parallel 1 \
  --use-pmc \

  --pmc-initialize-from-file mcmc_c10-B.hdf5 \
  --pmc-dof -1 \
  --pmc-group-by-r-value 1.5 \
  --hc-target-ncomponents 50 \                                                   PMC
  --hc-patch-length 300 \
  --hc-skip-initial 0.2 \
  --pmc-samples-per-component 3000 \
  --pmc-final-samples 100000 \
  --pmc-adjust-sample-size 1 \
  --pmc-relative-std-deviation-over-last-steps 0.2 2 \
  --pmc-ignore-ess 1 \

  --global-option model WilsonScan \                                             physics analysis
  --global-option scan-mode cartesian \
  --scan Re{c10} -6.0 +6.0 --prior flat \
  --nuisance CKM::lambda        3 --prior gaussian 0.2247 0.2253 0.2259 \
  --nuisance CKM::A             3 --prior gaussian 0.787 0.807 0.827 \
  --nuisance CKM::rhobar 0.0 1.0 3 --prior gaussian 0.073 0.128 0.183 \
  --nuisance CKM::etabar        3 --prior gaussian 0.315 0.375 0.435 \
  --nuisance decay-constant::B_s 3 --prior gaussian 0.2232 0.2277 0.2322 \
  --constraint "B^0_s->mu^+mu^-::BR@CMS-LHCb-2014" \

  --output pmc_monolithic_c10-B.hdf5                                             output
```

Example: C_{10} in $B_s \to \mu^+\mu^-$ — PMC runtime output

The current analysis configuration (same as MCMC)

```
# Scan generated by eos-scan-mc
# Scan parameters (1):
#   Parameter: Re{c10}, prior type: flat, range: [-6,6]
# Nuisance parameters (5):
#   Parameter: CKM::A, prior type: Gaussian, range: [0.747,0.867], x = 0.807 +- 0.02
#   Parameter: CKM::lambda, prior type: Gaussian, range: [0.2234,0.2273], x = 0.22535 +- 0.00065
#   Parameter: CKM::rhobar, prior type: Gaussian, range: [0,0.293], x = 0.128 +- 0.055
#   Parameter: CKM::etabar, prior type: Gaussian, range: [0.195,0.555], x = 0.375 +- 0.06
#   Parameter: decay-constant::B_s, prior type: Gaussian, range: [0.2126,0.2426], x = 0.2276 +- 0.005
# Constraints (1):
#  B^0_s->mu^+mu^-::BR@CMS-LHCb-2014: B_q->ll::BR@Untagged[] with options: l=mu,model=WilsonScan,q=s,scan
      -mode=cartesian, Amoroso limit: mode at B_q->ll::BR@Untagged = 2.8e-09 (a = 0, theta = 1.5424e-10,
      alpha = 19.402, beta = 1.0048)
...
```

- can verify correct settings for scan and nuisance parameters
- details on constraints

Example: C_{10} in $B_s \to \mu^+\mu^-$ — PMC runtime output

Info on PMC initialisation = hierarchical clustering

```
...
...: [INFO PMC_sampler::initialize] Reading from file mcmc_c10-B.hdf5
...: [DEBUG PMC.hierarchical_clustering] Added chain 1 to group 0
...: [DEBUG PMC.hierarchical_clustering] Added chain 2 to group 0
...: [DEBUG Cluster.overlaps] Parameter 0: r value too large (8.442166281 > 1.5)
...: [DEBUG PMC.hierarchical_clustering] Created new group for chain 3
...: [INFO PMC_sampler.hierarchical_clustering] Found 2 groups of chains with ( 3 1 ) members
...: [INFO PMC_sampler.hierarchical_clustering] Creating initial guess for the 50 target components to be
       formed from large windows for each of the 2 chain groups found
...: [INFO PMC_sampler.hierarchical_clustering] Creating patches of length 300
...: [INFO PMC_sampler.hierarchical_clustering] Formed 1064 input components centered around patch means
...: [INFO PMC_sampler.hierarchical_clustering] Start hierarchical clustering
...: [DEBUG HierarchicalClustering::run] Current distance in step 0: 0.4799140459544602
...: [DEBUG HierarchicalClustering::run] Current distance in step 1: 0.4474365922442486
...
...: [DEBUG HierarchicalClustering::run] Current distance in step 14: 0.4307752076516199
...: [INFO HierarchicalClustering::run] Found exact local minimum after 14 steps
```

- **group** the 4 Markov chains (MC) from MCMC pre-run via R-value `--pmc-group-by-r-value 1.5`
  $\Rightarrow$ in example two groups found: one with 3 MC's, one with 1 MC
- for each group 50 **target components** are created, `--hc-target-ncomponents 50`
  $\Rightarrow$ in total $2 \times 50 = 100$ to be used in PMC run
- chop MC's into patches of 300 samples and create components (in total 1064)
  `--hc-patch-length 300`
- use hierarchical clustering to "map" 1064 components on 100 target components
  $\Rightarrow$ information of MC's of MCMC pre-run transformed in initial components of PMC

Example: C_{10} in $B_s \to \mu^+\mu^-$ — PMC runtime output

and PMC Main-run

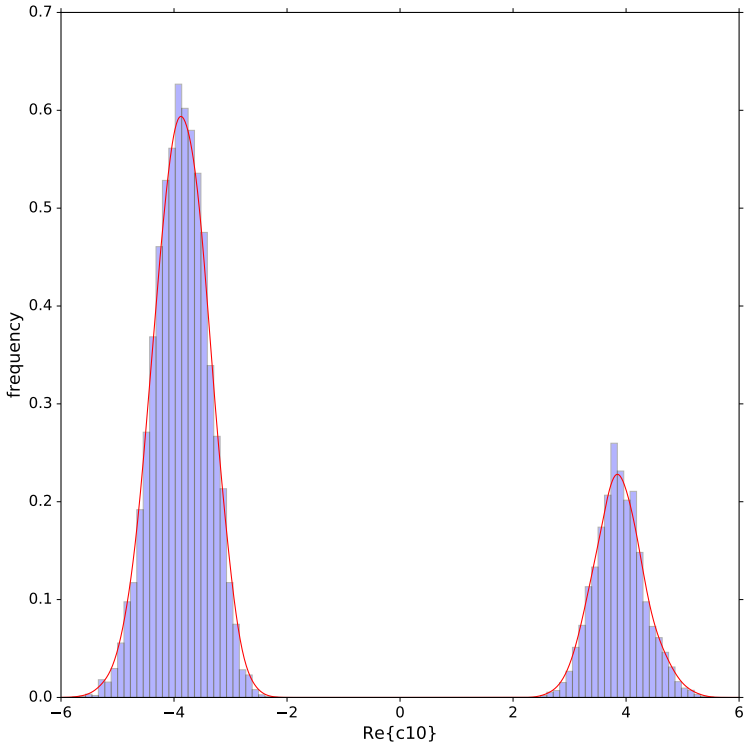
```
...: [DEBUG PMC_sampler.status] Drawing samples
...: [DEBUG PMC_sampler.status] Calculating 300000 samples
...
...: [INFO PMC_sampler.status] Updating the proposal function
normalize_importance_weight: isLog=0, returning 31.1786
...: [INFO PMC_sampler.status] Status after step 1 of 10 with 300000 samples:
...: [INFO PMC_sampler.status] perplexity: 0.9921232443, eff. sample size: 0.9839676509, evidence:
      115767474.2
...: [INFO PMC_sampler.check_convergence] perplexity = 0.9921, effective sample size = 0.984
...: [DEBUG PMC_sampler.check_convergence] perplexity (0.9921) large enough
...: [INFO PMC_sampler.status] Convergence achieved after 1 steps.
...
normalize_importance_weight: isLog=0, returning 30.0786
...: [INFO PMC_sampler.status] Status after final step with 100000 samples:
...: [INFO PMC_sampler.status] perplexity: 0.9951256156, eff. sample size: 0.9902980771, evidence:
      115607520.7
...: [INFO PMC_sampler.dump] 0 out of 100 components died out.
```

- **drawing samples** from target components = proposal function
- **calculating likelihood** for samples = (number of components) × **`(--pmc-samples-per-component 3000)`**
- **updating components** with new samples using PMC-algorithm
- **checking convergence** criteria: perplexity (optional ESS) ⇒ convergence after 1 step
- use proposal function to calculate final samples **`--pmc-final-samples 100000`**
- exits after main-run with generation of output file **`"pmc_monolithic_c10-B.hdf5"`**

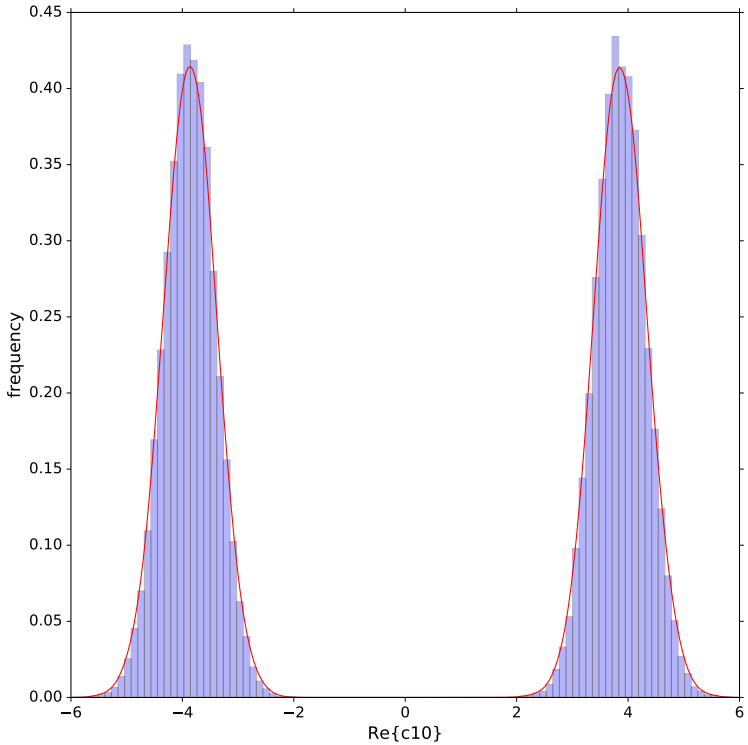
Example: C_{10} in $B_s \to \mu^+\mu^-$ — plot

Make a plot of the result

```
> python ../scripts/eos-plot-1d pmc_monolithic_c10-B.hdf5 0 pmc_c10-B_0.pdf
```



MCMC — wrong weights



PMC — correct weights

PMC gets correct posterior distributions for multi-modal target densities

MCMC serves as a good initialisation

Both in EOS

Uncertainty propagation

Example: Posterior predictive for $B_d \to \mu^+\mu^-$

Uncertainty propagation: from prior- or posterior-samples of parameters

$\Rightarrow$ determine **prior- and posterior-predictives of observables**

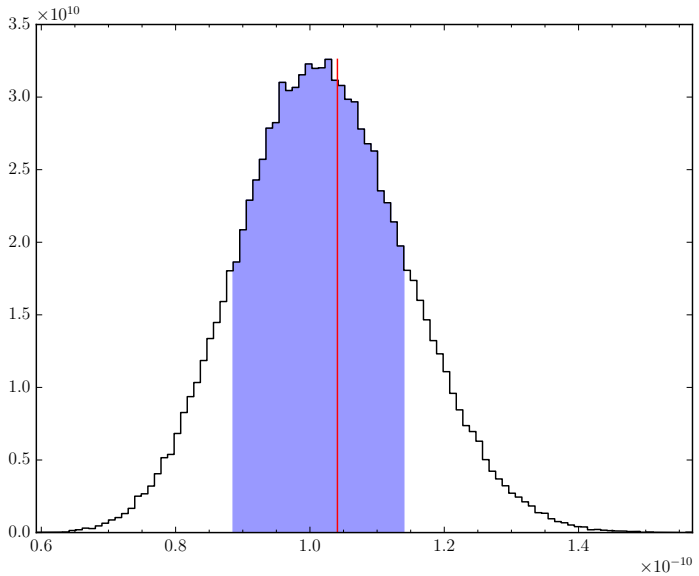
- let's use the final samples from the fit of C_{10} from $Br(B_s \to \bar{\mu}\mu)$ to make a posterior-predictive of $Br(B_d \to \bar{\mu}\mu)$

$\Rightarrow$ assumes that new physics is minimal flavour violating, i.e. C_{10} is same in both decays

- can be done with **EOS**-client `"eos-propagate-uncertainty"`

```
> eos-propagate-uncertainty \
    --observable "B_q->ll::BR,model=WilsonScan,q=d" \
    --pmc-input pmc_monolithic_c10-B.hdf5 0 100000 \
    --pmc-sample-directory '/data/final' \
    --output unc_c10-B.hdf5
```

- declare observable(s)
- specify input file of samples and HDF5-directory within
- specify output file for samples of observables
- can provide private python script to generate the plot



$Br(B_d \to \bar{\mu}\mu) \times 10^{-10}$

Other Use Cases

- **Goodness-of-fit analysis**: find best fit point, calculate p value, etc.
- **Sampling events from signal-PDF** with client **`eos-sample-events-mcmc`**
  (needs to be documented, approach Danny if you are interested)
  for implemented decays: $B \to D\mu\nu_\mu$, $B \to D\tau\nu_\tau$, $B \to K^{(*)}\bar{\mu}\mu$

Backup Slides