Technische Universität München

Flavour Physics with EOS

13.06.2019

Danny van Dyk

Technische Universität München

funded by DFG Deutsche Forschungsgemeinschaft

Emmy Noether-Programm DFG Deutsche Forschungsgemeinschaft

What is EOS

What is EOS

software framework for applications in high-energy physics; in particular flavour physics

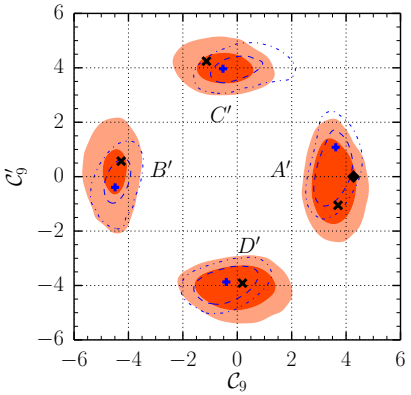
- written in C++14 with an extensive test suite
- can be used via both a C++14 and a Python3 API
  - using EOS via Jupyter notebooks recommended for new users
  - Python3 API documented online
- contributors

  theory DvD, Christoph Bobeth, Frederik Beaujean, Ahmet Kokulu, Nico Gubernari, Stephan Kürten, Marzia Bordone, Keri Vos, ...

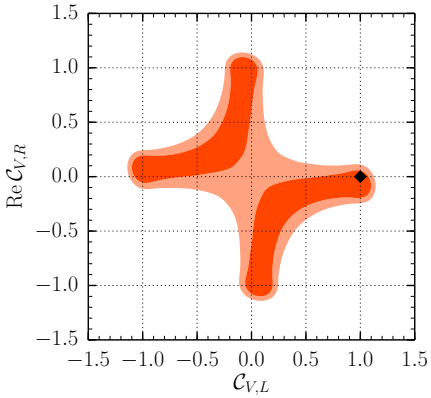
  experiment Thomas Blake, Christoph Langenbruch, Martin Ritter, ...

Track Record

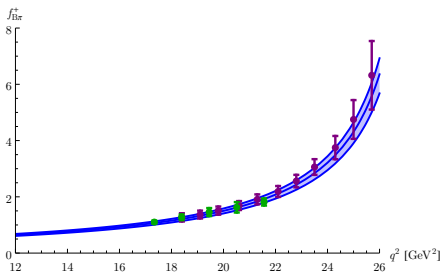
- first open source code for calculating observables in the $b \to s\ell\ell$ anomalies (first published 2011)
- developed within the Dortmund and Siegen groups, with substantial help from Munich
- used for 23+ peer-reviewed pheno publications
- used internally in 3+ LHCb analyses
- part of the “Belle-2 Externals” software library



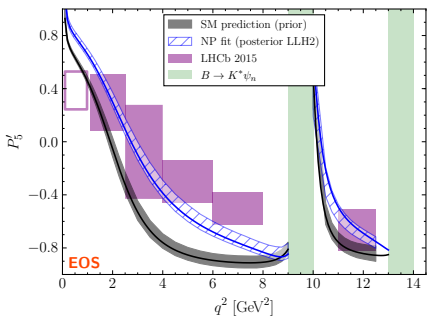
$b \to s\mu^+\mu^-$



$b \to u\mu^-\overline{\nu}$



$B \to \pi$ FF



$B \to K^*$ NL

Use Cases

EOS has been authored with three use cases in mind:

1. theory predictions and uncertainty estimates of flavour observables and related theoretical quantities
   - in the SM, and for arbitrary WET/LEFT inputs
   - “everything is a parameter”: both NP and hadronic parameters can be modified
   - aspires to produce theory / uncertainty estimates of publication quality

Use Cases

EOS has been authored with three use cases in mind:

1. theory predictions and uncertainty estimates of flavour observables and related theoretical quantities
   - in the SM, and for arbitrary WET/LEFT inputs
   - “everything is a parameter”: both NP and hadronic parameters can be modified
   - aspires to produce theory / uncertainty estimates of publication quality
2. parameter Inference from experimental measurements and/or theoretical constraints.
   - defaults to the Bayesian framework of parameter inference
   - provides a database of experimental measurements and theoretical constraints for immediate use

Use Cases

EOS has been authored with three use cases in mind:

1. theory predictions and uncertainty estimates of flavour observables and related theoretical quantities
   - in the SM, and for arbitrary WET/LEFT inputs
   - "everything is a parameter": both NP and hadronic parameters can be modified
   - aspires to produce theory / uncertainty estimates of publication quality
2. parameter Inference from experimental measurements and/or theoretical constraints.
   - defaults to the Bayesian framework of parameter inference
   - provides a database of experimental measurements and theoretical constraints for immediate use
3. production of pseudo events for flavor-physics processes
   - in the SM, and for arbitrary WET/LEFT inputs

Physics Cases

out of the box

- exclusive $b \to s\{\ell\ell, \gamma\}$
  - mesons & baryons
- exclusive $b \to c\ell\overline{\nu}$
  - mesons & baryons
- exclusive $b \to u\ell\overline{\nu}$

work in progress

- $e^+e^- \to$ open charm
- $D \to \{\pi, K\}\ell\overline{\nu}$
- hadronic B decays

overview full list of processes and their observables:
`https://eos.github.io/doc/observables.html`

focus accurate desription and versatility in parametrization/calculation of hadronic matrix elements

Availability

- easy installation on Ubuntu Linux and MacOS X
- Ubuntu: binary packages

```
DIST=$(lsb_release -c)
echo deb https://packagecloud.io/eos/eos/ubuntu/ $DIST main \
    | sudo tee /etc/apt/sources.list.d/eos.list
curl -L "https://packagecloud.io/eos/eos/gpgkey" 2> /dev/null \
    | sudo apt-key add - &>/dev/null
sudo apt-get update
sudo apt-get install eos
```

- MacOS X: building from source using Homebrew

```
brew tap eos/eos
brew install --HEAD eos
pip3 install --user h5py matplotlib numpy scipy
```

- other OS: need installation from source; see me after the talk

Examples

About the Examples / First Steps

- only showing most salient parts of the examples
- full example notebooks available online
  - evaluating observables / estimating theory uncertainties
  - infering parameters
  - interfacing with WCxf / Wilson
- will assume that you make EOS available to your Python interpreter

```
1  import eos
```

Listing the built-in Observables

List all branching ratios with a D meson in the final state

```
1  eos.Observables(prefix='->D', name='BR')
```

Observables in (semi)leptonic b-hadron decays

Observables in $B \to \bar{D}\ell^-\bar{\nu}$ decays

B->Dlnu::BR	$\mathcal{B}(B \to \bar{D}\ell^-\bar{\nu})$
B->Dlnu::dBR/dq2	$d\mathcal{B}(B \to \bar{D}\ell^-\bar{\nu})/dq^2$

The option "l" selects the charged lepton flavour. The option "q" selects the spectator quark flavour. The option "form-factors" selects the form factor parametrization.

Observables in $B \to \bar{D}^*\ell^-\bar{\nu}$ decays

B->D^*lnu::BR	$\mathcal{B}(B \to \bar{D}^*\ell^-\bar{\nu})$
B->D^*lnu::dBR/dq2	$d\mathcal{B}(B \to \bar{D}^*\ell^-\bar{\nu})/dq^2$

The option "l" selects the charged lepton flavour. The option "q" selects the spectator quark flavour. The option "form-factors" selects the form factor parametrization.

B->Dlnu::dBR/dq2

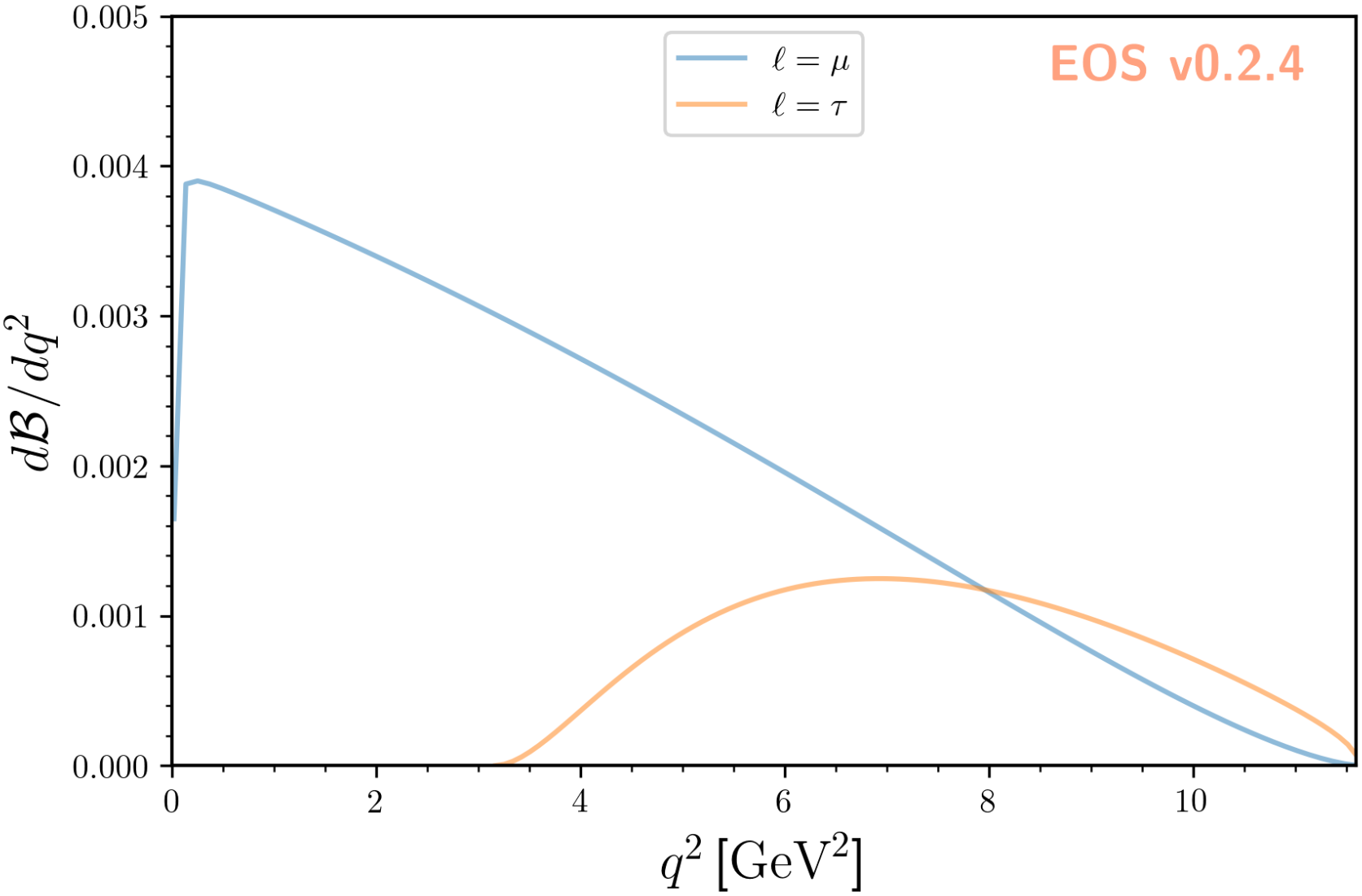
Plotting the central values

Create a plot of built-in observables with default Parameters

```
plot_args = {
  'plot': {
    'x': { 'label': r'$q^2$', 'unit': r'$\textnormal{GeV}^2$',
           'range': [0.0, 11.60]                                  },
    'y': { 'label': r'$d\mathcal{B}/dq^2$',
           'range': [0.0,  5e-3]                                  },
    'legend': { 'location': 'upper center' }
  },
  'contents': [
    { 'label': r'$\ell=\mu$',  'type': 'observable',
      'observable': 'B->Dlnu::dBR/dq2;l=mu',
      'kinematic': 'q2', 'range': [0.02, 11.60], },
    { 'label': r'$\ell=\tau$', 'type': 'observable',
      'observable': 'B->Dlnu::dBR/dq2;l=tau',
      'kinematic': 'q2', 'range': [3.17, 11.60], }
  ]
}
eos.plot.Plotter(plot_args).plot()
```

Plotting the central values

Plot built-in observables with default Parameters



Set up an analysis with hadronic nuisance parameters from the Lattice

```
analysis_args = {
    'global_options': None,
    'priors': [
        { 'parameter': 'B->D::alpha^f+_0@BSZ2015', 'min':  0.0, 'max':  1.0 },
        { 'parameter': 'B->D::alpha^f+_1@BSZ2015', 'min': -5.0, 'max': +5.0 },
        { 'parameter': 'B->D::alpha^f+_2@BSZ2015', 'min': -5.0, 'max': +5.0 },
        { 'parameter': 'B->D::alpha^f0_1@BSZ2015', 'min': -5.0, 'max': +5.0 },
        { 'parameter': 'B->D::alpha^f0_2@BSZ2015', 'min': -5.0, 'max': +5.0 }
    ],
    'likelihood': [
        'B->D::f_++f_0@HPQCD2015A',
        'B->D::f_++f_0@FNALMILC2015A'
    ]
}
analysis = eos.Analysis(**analysis_args)
```

Produce samples for the observables

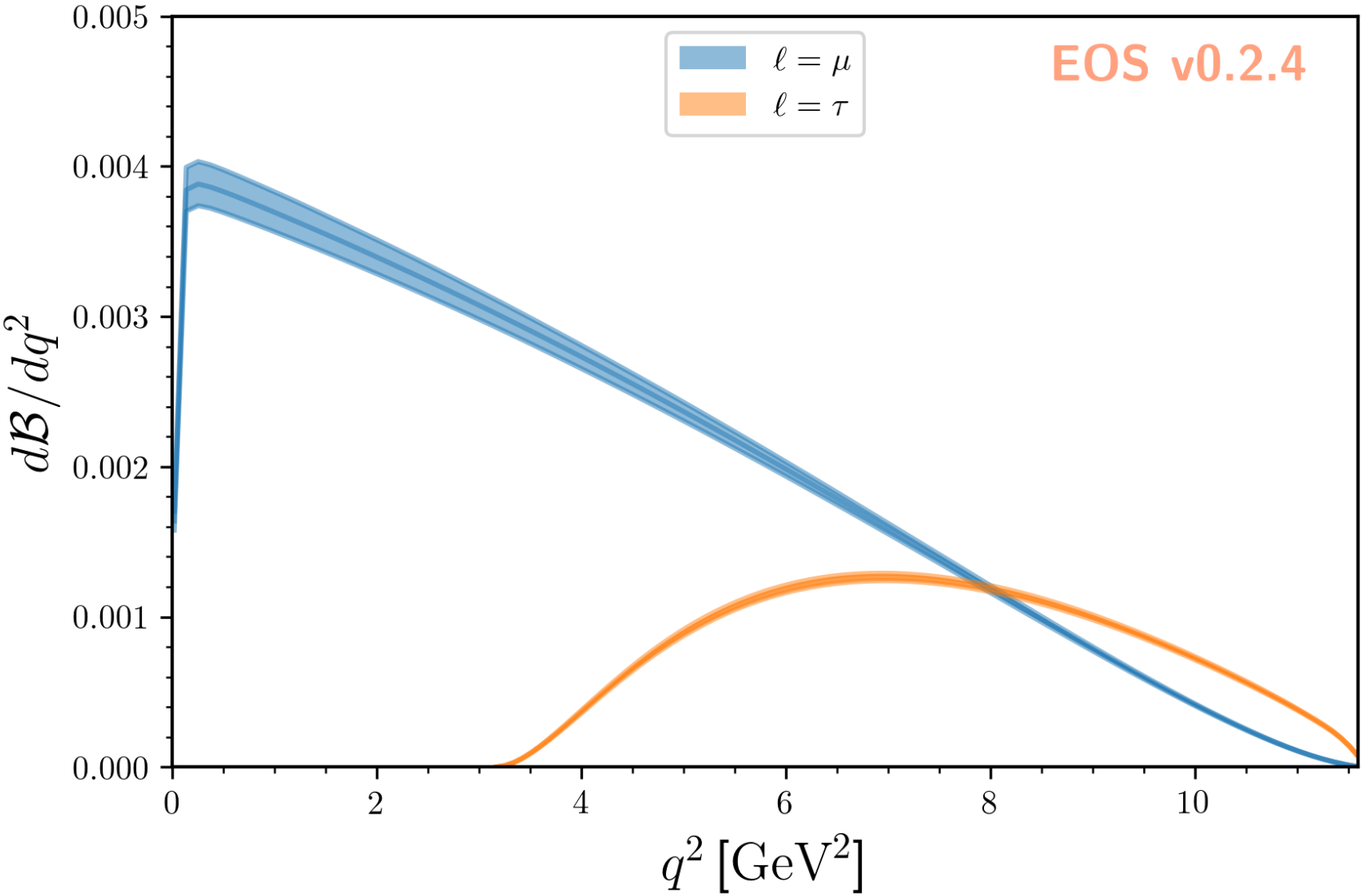
```
mu_q2values = np.unique(
        np.concatenate((np.linspace(0.02,  1.00, 20),
        np.linspace(1.00, 11.60, 20)))
)
mu_obs      = [eos.Observable.make(
                'B->Dlnu::dBR/dq2', analysis.parameters,
                eos.Kinematics(q2=q2),
                eos.Options(**{'form-factors': 'BSZ2015', 'l': 'mu'}))
              for q2 in mu_q2values]
tau_q2values= np.linspace(3.17, 11.60, 40)
tau_obs     = [eos.Observable.make(
                'B->Dlnu::dBR/dq2', analysis.parameters,
                eos.Kinematics(q2=q2),
                eos.Options(**{'form-factors': 'BSZ2015', 'l': 'tau'}))
              for q2 in tau_q2values]

_, log_weights, mu_samples  = analysis.sample(pre_N=1000, observables=mu_obs)
_, log_weights, tau_samples = analysis.sample(pre_N=1000, observables=tau_obs)
```

Plot bands with 68%-probability envelope

```
plot_args = {
  'plot': {
    'x': { 'label': r'$q^2$', 'unit': r'$\textnormal{GeV}^2$',
           'range': [0.0, 11.60]                                  },
    'y': { 'label': r'$d\mathcal{B}/dq^2$',
           'range': [0.0,  5e-3]                                  },
    'legend': { 'location': 'upper center' }
  },
  'contents': [
    { 'label': r'$\ell=\mu$', 'type': 'uncertainty',
      'range': [0.02, 11.60],
      'data': { 'samples': mu_samples, 'xvalues': mu_q2values }
     },
    { 'label': r'$\ell=\tau$','type': 'uncertainty',
      'range': [3.17, 11.60],
      'data': { 'samples': tau_samples, 'xvalues': tau_q2values }
     },
  ]
}
eos.plot.Plotter(plot_args).plot()
```

Plot bands with 68%-probability envelope



List all experimental constraints for branching ratios with a D meson in the final state

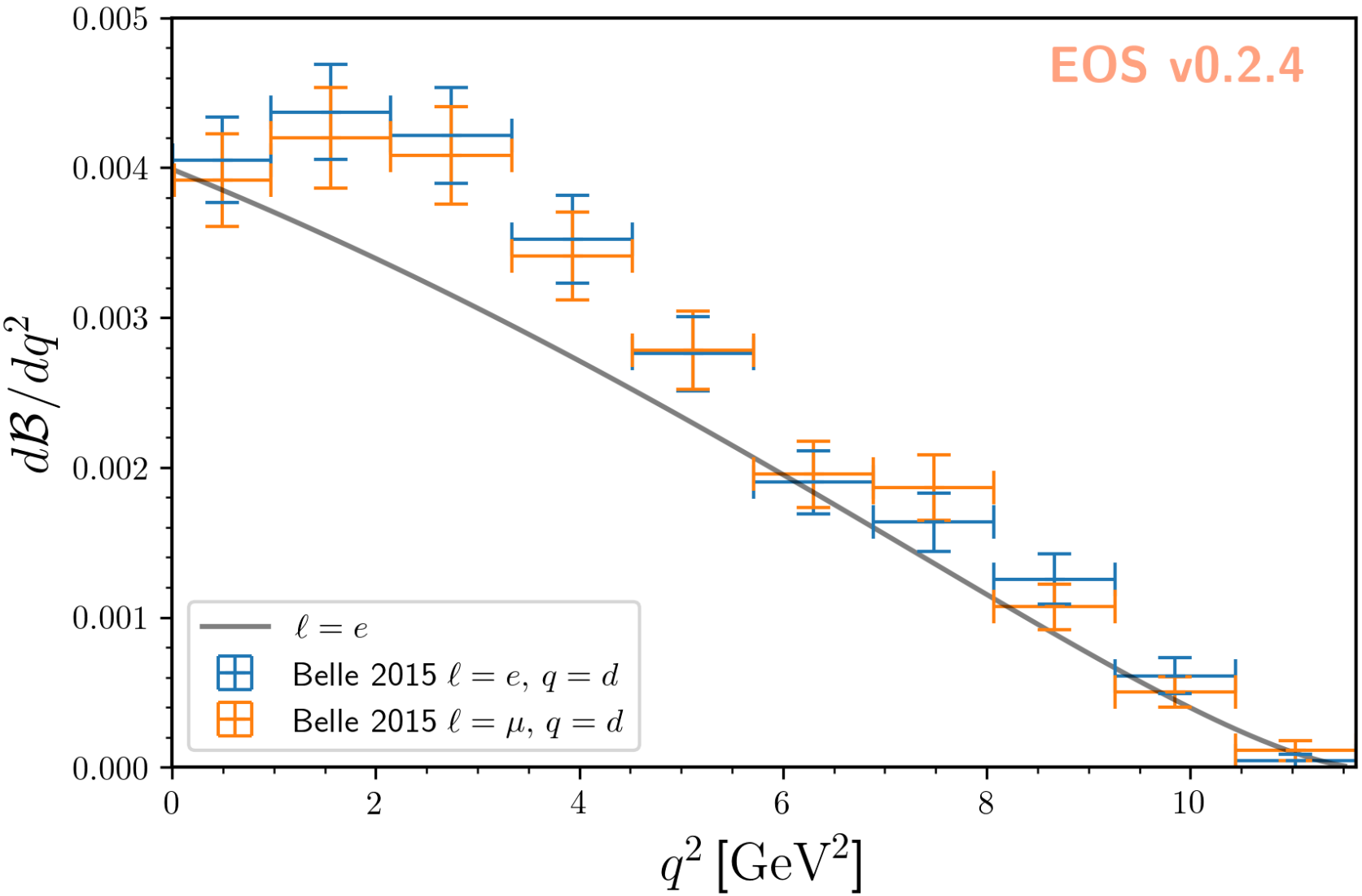
```
1  eos.Constraints(prefix='->D', name='BR')
```

Name	Type
B^0->D^+e^-nu::BRs@Belle-2015A	MultivariateGaussian<33> (using covariance matrix)
B^0->D^+mu^-nu::BRs@Belle-2015A	MultivariateGaussian<33> (using covariance matrix)

Plot default prediction and illustrate exp. constraints

```
plot_args = {
   # [ plot setup as in earlier plotting example ... ]
  'contents': [
    # [ SM plot cut...]
    { 'label': r'Belle 2015 $\ell=e,\, q=d$', 'color': 'C0',
      'type': 'constraint',
      'constraints': 'B^0->D^+e^-nu::BRs@Belle-2015A',
      'observable': 'B->Dlnu::BR', 'variable': 'q2',
      'rescale-by-width': False                                    },
    { 'label': r'Belle 2015 $\ell=\mu,\,q=d$', 'color': 'C1',
      'type': 'constraint',
      'constraints': 'B^0->D^+mu^-nu::BRs@Belle-2015A',
      'observable': 'B->Dlnu::BR', 'variable': 'q2',
      'rescale-by-width': False                                    },
  ]
}
eos.plot.Plotter(plot_args).plot()
```

Plot default prediction and illustrate exp. constraints



Set up analysis as before plus Belle data on $\overline{B} \to D\{e^-, \mu^-\}\overline{\nu}$

```
analysis_args = {
  'global_options': { 'form-factors': 'BSZ2015', 'model': 'CKMScan' },
  'priors': [
    { 'parameter': 'CKM::abs(V_cb)', 'min':  38e-3, 'max':  48e-3  },
    # [ same hadronic parameters as in uncertainty estimates ... ]
  ],
  'likelihood': [
    'B->D::f_++f_0@HPQCD2015A',       # theory
    'B->D::f_++f_0@FNALMILC2015A',    # theory
    'B^0->D^+e^-nu::BRs@Belle-2015A', # experiment
    'B^0->D^+mu^-nu::BRs@Belle-2015A' # experiment
  ]
}
analysis = eos.Analysis(**analysis_args)
```

Find and display best-fit point and goodness-of-fit summary

Produce random samples of the posterior

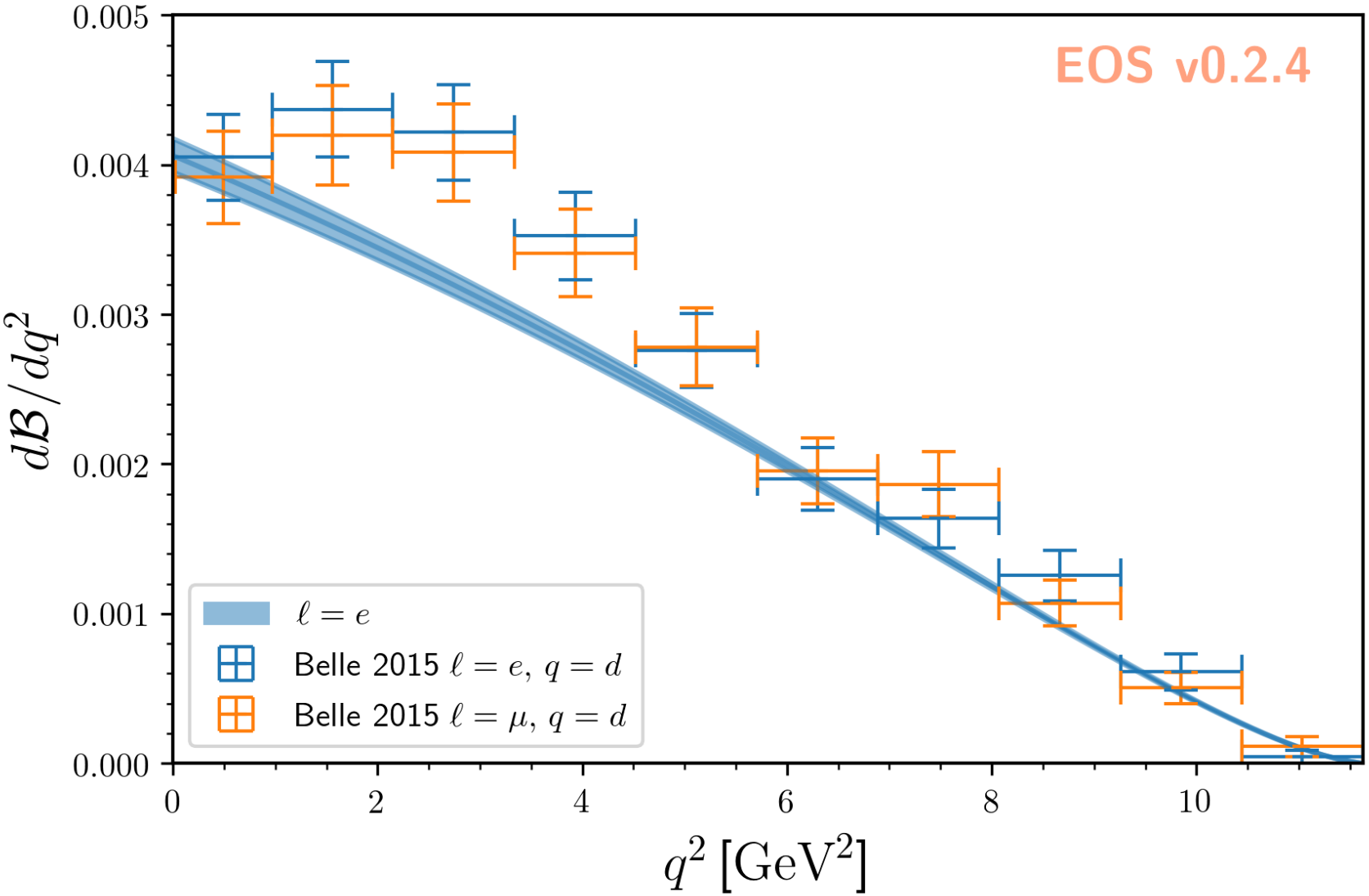
```
bfp = analysis.optimize()
display(bfp, analysis.goodness_of_fit())
parameter_samples, log_weights = analysis.sample(
    N=20000, stride=5, pre_N=1000, preruns=5,
    cov_scale=0.05, start_point=bfp.point)
```

parameter	value
$\lvert V_{cb}\rvert$	0.0422
B->D::alpha^f+_0@BSZ2015	0.6670
B->D::alpha^f+_1@BSZ2015	-2.5344
B->D::alpha^f+_2@BSZ2015	4.8537
B->D::alpha^f0_1@BSZ2015	0.2633
B->D::alpha^f0_2@BSZ2015	-0.8655

constraint	χ^2	d.o.f.
B->D::f_++f_0@FNALMILC2015A	3.4933	7
B->D::f_++f_0@HPQCD2015A	3.0796	5
B^0->D^+e^-nu::BRs@Belle-2015A	11.8283	10
B^0->D^+mu^-nu::BRs@Belle-2015A	5.2287	10

total χ^2	23.6300
total degrees of freedom	26
p-value	59.7114%

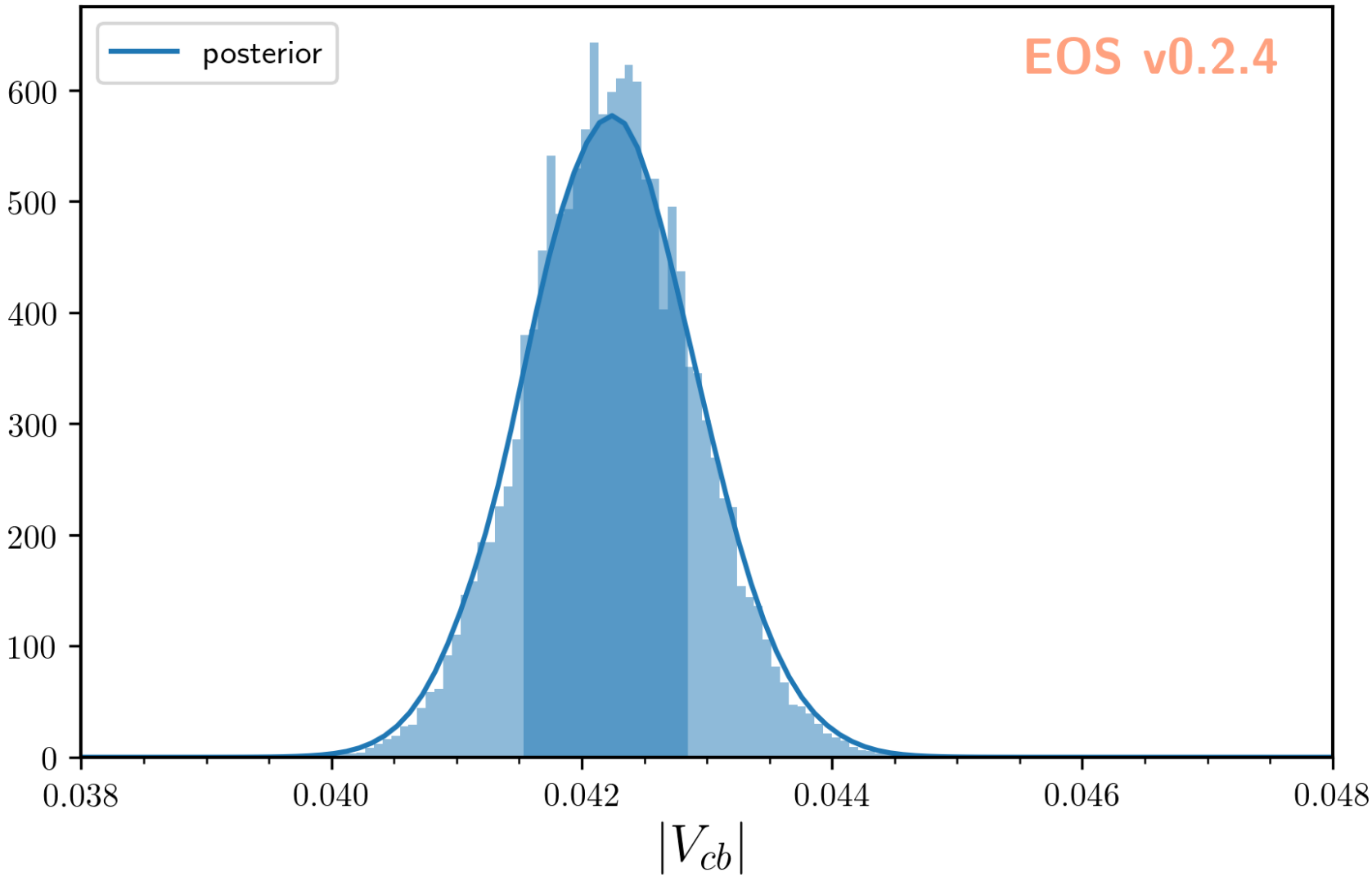
Plot posterior-predictive distribution



Plot random samples of $|V_{cb}|$

```
plot_args = {
  'plot': {
    'x': { 'label': r'$|V_{cb}|$', 'range': [38e-3, 48e-3] },
    'legend': { 'location': 'upper left' }
  },
  'contents': [
    {
      'type': 'histogram',
      'data': { 'samples': parameter_samples[:, 0], 'log_weights': log_weights
      }
    },
    {
      'type': 'kde', 'color': 'C0', 'label': 'posterior', 'bandwidth': 2,
      'range': [40e-3, 45e-3],
      'data': { 'samples': parameter_samples[:, 0], 'log_weights': log_weights
      }
    }
  ]
}
eos.plot.Plotter(plot_args).plot()
```

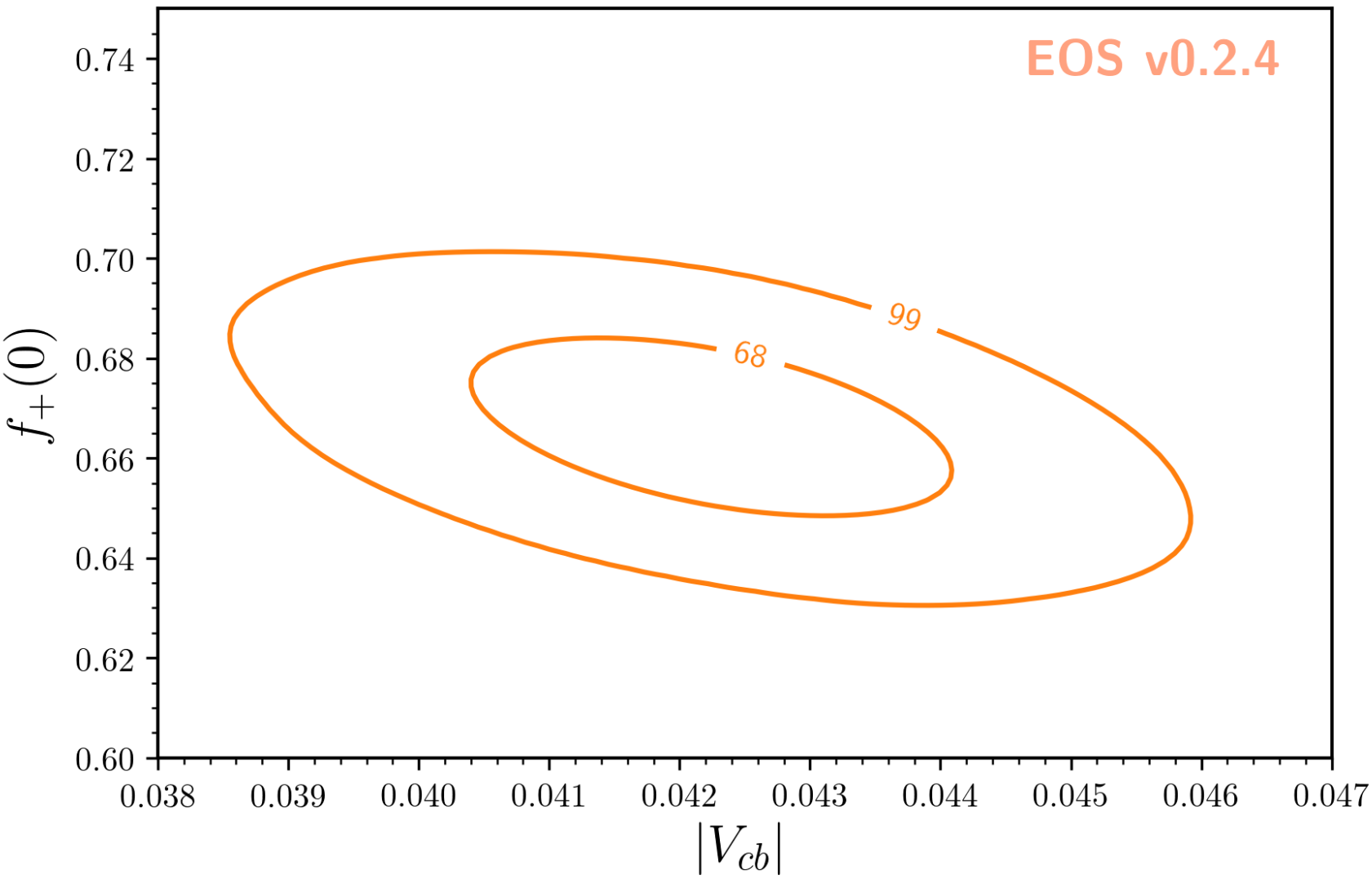
Plot random samples of $|V_{cb}|$



Plot random samples of $|V_{cb}|$ vs $f_+(q^2 = 0)$

```
plot_args = {
  'plot': {
    'x': { 'label': r'$|V_{cb}|$', 'range': [38e-3, 47e-3] },
    'y': { 'label': r'$f_+(0)$', 'range': [0.6, 0.75] },
  },
  'contents': [
    {
      'type': 'kde2D', 'color': 'C1', 'label': 'posterior',
      'range': [40e-3, 45e-3], 'levels': [68, 99], 'bandwidth': 3,
      'data': { 'samples': parameter_samples[:, (0,1)],
                'log_weights': log_weights                }
    }
  ]
}
eos.plot.Plotter(plot_args).plot()
```

Plot random samples of $|V_{cb}|$ vs $f_+(q^2 = 0)$



Wilson / WCxf

Create SMEFT parameter point, match to WET, and run to 4.2 GeV

```python
from wilson import Wilson
import numpy as np
wilson_in = Wilson({
    'lq3_2223': +1.10e-9,
    'phiq3_23': +1.10e-9
}, scale = 1e3, eft = 'SMEFT', basis = 'Warsaw')
wcxf_out  = wilson_in.match_run(scale=4.2, eft='WET', basis='EOS')
```

with big thanks to Jason Aebischer and David Straub!

Wilson / WCxf

Create EOS Observables for the SM and NP benchmark point and evaluate

```
import eos
p_SM = eos.Parameters.Defaults()
p_NP = eos.Parameters.FromWCxf(wcxf_out)

o = eos.Options(model='WilsonScan')
k = eos.Kinematics(q2_min=1,q2_max=6)
obs_RK_SM   = eos.Observable.make('B->Kll::R_K@LargeRecoil',      p_SM, k, o)
obs_RKst_SM = eos.Observable.make('B->K^*ll::R_K^*@LargeRecoil', p_SM, k, o)
obs_RK_NP   = eos.Observable.make('B->Kll::R_K@LargeRecoil',      p_NP, k, o)
obs_RKst_NP = eos.Observable.make('B->K^*ll::R_K^*@LargeRecoil', p_NP, k, o)

display(obs_RK_SM)
display(obs_RKst_SM)
display(obs_RK_NP)
display(obs_RKst_NP)
```

Create EOS Observables for the SM and NP benchmark point and evaluate

B->Kll::R_K@LargeRecoil (eos.Observable)

current value	1.001

B->K^*ll::R_K^*@LargeRecoil (eos.Observable)

current value	0.9988

B->Kll::R_K@LargeRecoil (eos.Observable)

current value	0.7325

B->K^*ll::R_K^*@LargeRecoil (eos.Observable)

current value	0.7603

Conclusion

Conclusion

Summary:

- EOS is a powerful tool for flavour-physics applications
- focus on hadronic matrix elements, their parametrizations and parameters
- easy-to-use Python3 API + computationally efficient C++-14 “backend”

Not shown today:

- production of pseudo events
- extending EOS with new observables, new parameters, new constraints

  → see the manual and online documentation
- command-line interface for use on headless computers / clusters
- more powerful statistical algorithms (Population Monte Carlo) for inference of $\mathcal{O}(100)$ parameters