PyHF use-case: Branching Ratio Determination of $B \to D^{**}\ell\nu$

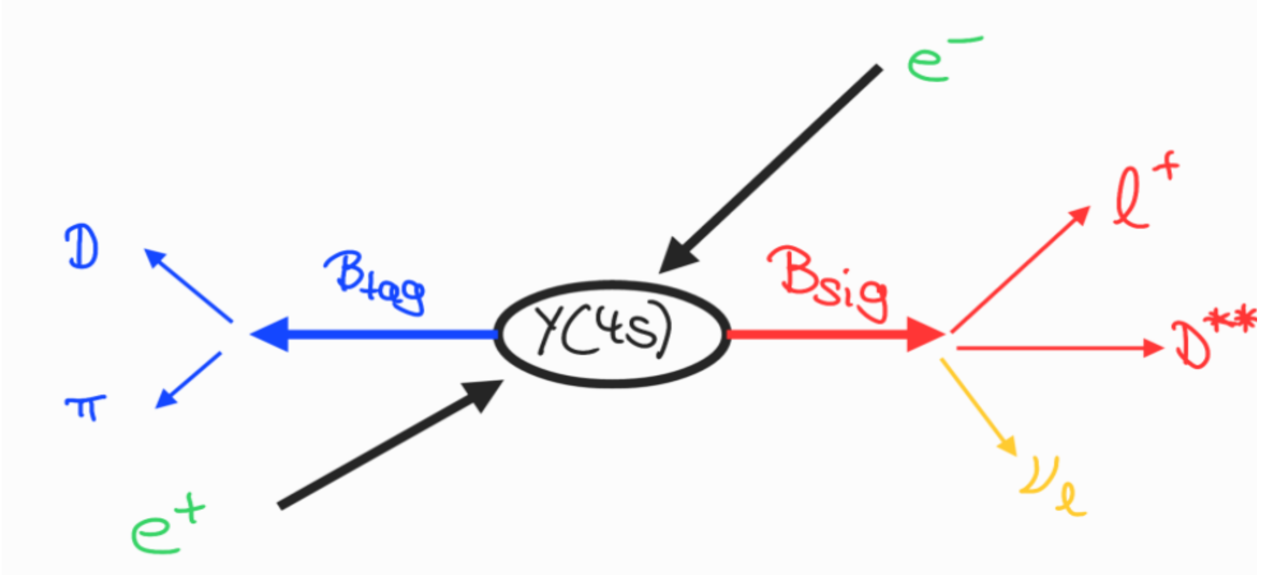
Noreen Rauls

(Universität Göttingen)

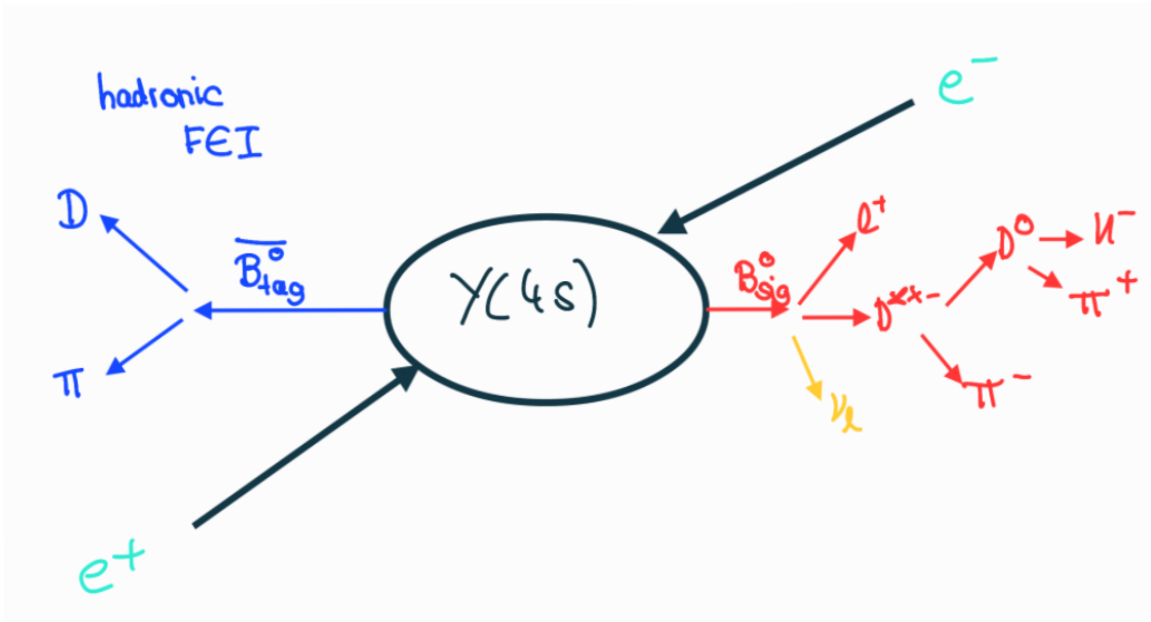
Analysis Goal

- measurement of branching ratio $B \to D^{**} \ell \nu$ with hadronic FEI
- normalised to $B \to D^{*} \ell \nu$

  → in total two fits



Analysis Overview

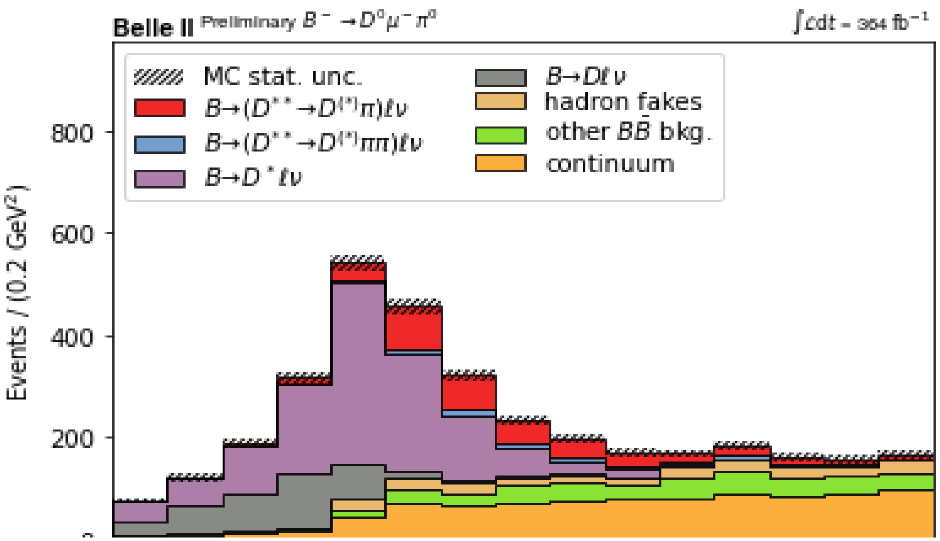


Decay channel
Neutral
$B^0 \to D^- \pi^0 \ell^+$
$B^0 \to D^{*-} \pi^0 \ell^+$
$B^0 \to \overline{D}^0 \pi^- \ell^+$
$B^0 \to \overline{D}^{*0} \pi^- \ell^+$
Charged
$B^- \to D^0 \pi^0 \ell^-$
$B^- \to D^{*0} \pi^0 \ell^-$
$B^- \to D^+ \pi^- \ell^-$
$B^- \to D^{*+} \pi^- \ell^-$

Neutral and charged B_{Sig} decay channels

General Fit Setup

- do binned maximum likelihood fit in with bins in range



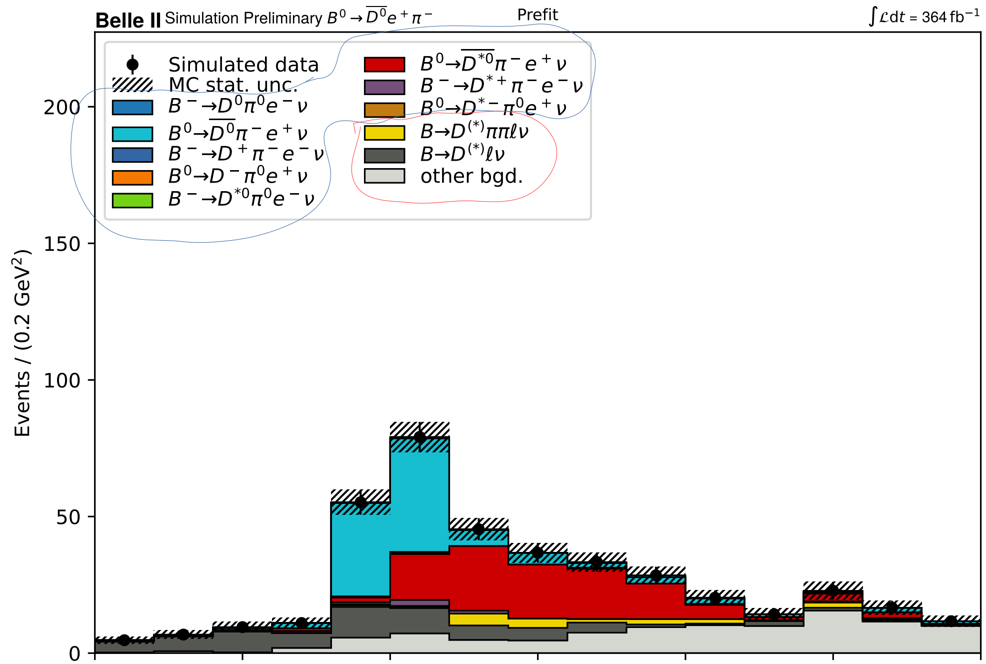
- combine backgrounds due to similar shapes
- divide signal into individual decay channels

General Fit Setup

- combine backgrounds due to similar shapes

  → 8 signal categories for D**

  → 3 background categories

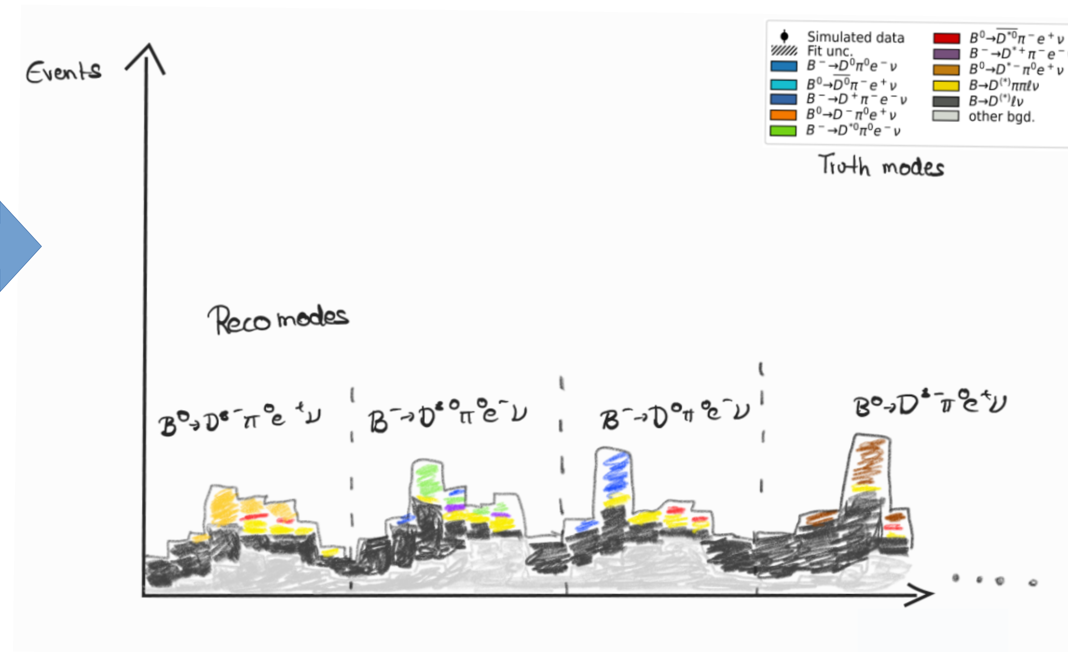


General Fit Setup

- to incorporate between overlap between D** reconstruction modes

  → one overall fit for each lepton mode

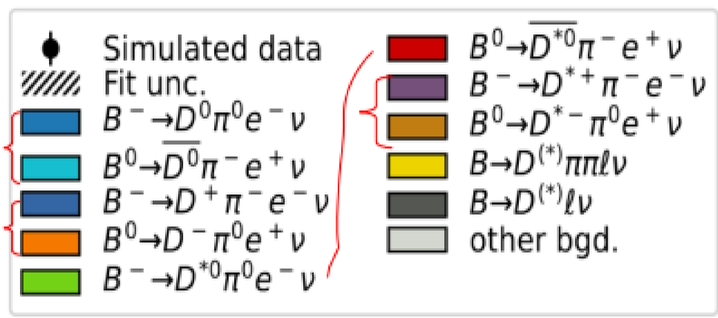
  → e and mu



General Fit Setup

- include isospin constraints

- Same scaling parameter




```
"modifiers": [
  {
    "name": "mu",
    "type": "normfactor",
    "data": null
  }
]
```

Systematic Uncertainties

- included systematic uncertainties as nuisance parameters
  - LeptonID, pi0, HadronID, Tracking, slow pi, photon, lumi

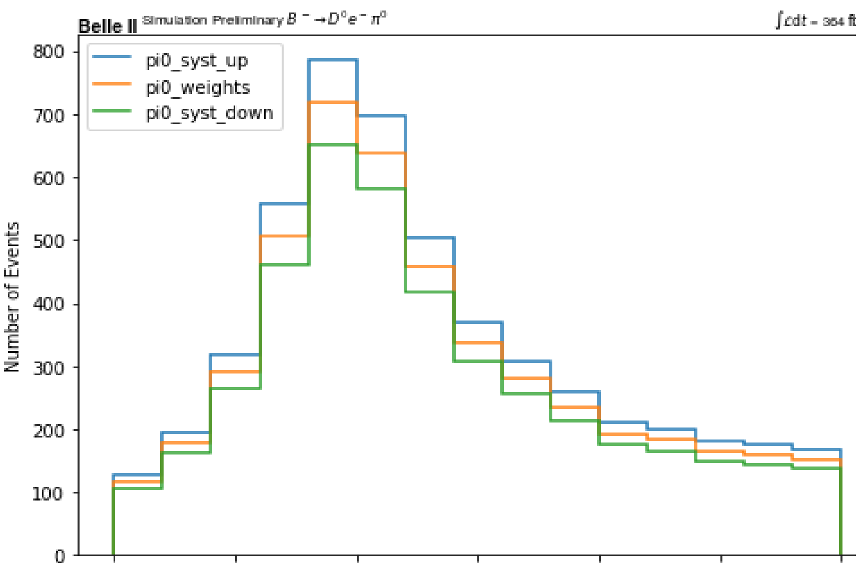
- for lumi include as parameter

```
"parameters": [
    {
        "name": "lumi",
        "auxdata": [1.0],
        "sigmas": [0.017],
        "bounds": [[0.915, 1.085]],
        "inits": [1.0],
    }
],
```

- Statistical uncertainties from MC

```
"modifiers": [
    {
        "name": "my_staterror",
        "type": "staterror",
        "data": [1.0, 2.0],
    }
],
```

Systematic Uncertainties




```
"modifiers": [
    {
        "name": "my_histosys",
        "type": "histosys",
        "data": {"hi_data": [15, 22], "lo_data": [5, 18]},
    }
],
```

- per-bin shape uncertainty controlled by single nuisance parameter (gaussian constraint)

  → in Belle II: most systematic uncertainties consist of stat. and syst. component

My Model

{'channels': [{'name': 'measurement', 'samples': [{'name': 'other bgd.', 'data': [0.5970679330901616, 1.3137996892535455, 1.2717256755698056, 1.4321599587267346, 4.768404764908382, 5.
802474698527642, 8.41263112225653, 14.083063154219614, 12.059218291109513, 16.644311234422375, 16.06636699396259, 21.543218196398243, 21.229423218210467, 30.185717718535496, 28.291736
452224203, 1.280398953542008, 0.9052616897885521, 2.5445884915542245, 6.599197818761798, 10.32428133270756, 10.113429188529286, 21.018721696285162, 22.328896956071873, 25.808460073203

689002506125, 7.485574909292409, 0.8328054919872592, 2.065439445605206, 3.1042849437953195, 2.4393950035188037, 9.242139573325845, 11.765912298734579, 14.599793458945486, 26.807783250
906894, 31.581027224451447, 33.451972700520635, 37.98349213784194, 48.789304110211305, 46.287658233786104, 47.589673673148646, 58.63434209920508, 0.19856388596687488, 0.67170740678999
33, 0.23697402490247435, 1.6462592177424176, 4.796659720924173, 6.011007572753977, 4.111711923501238, 4.010731358721076, 6.195699190887513, 7.915145156440278, 8.470084053172302, 8.506

- Dictionary writing for model automated → many entries

Running the Fit

```python
def fit(
    observation: np.ndarray,
    model: pyhf.pdf.Model,
    minimiser: str,
    par_bounds: List = None,
    init_pars: Union[np.ndarray, None] = None,
):
    """
    Do not want that the program exists, if the fit failed.
    """
    # set the minimiser type, initial guess with scipy, after that use minuit, so errors included
    pyhf.set_backend("numpy", minimiser)
    if par_bounds is None:
        par_bounds = model.config.suggested_bounds()
    try:
        result = pyhf.infer.mle.fit(
            data=observation,
            pdf=model,
            par_bounds=par_bounds,
            init_pars=init_pars,
            return_uncertainties=True,
            maxiter=int(1e9),
            return_result_obj=True,
        )
        return result
    except Exception:
        pass
```

- possible to use minuit or scipy for minimising the likelihood

→ only minuit returns uncertainties

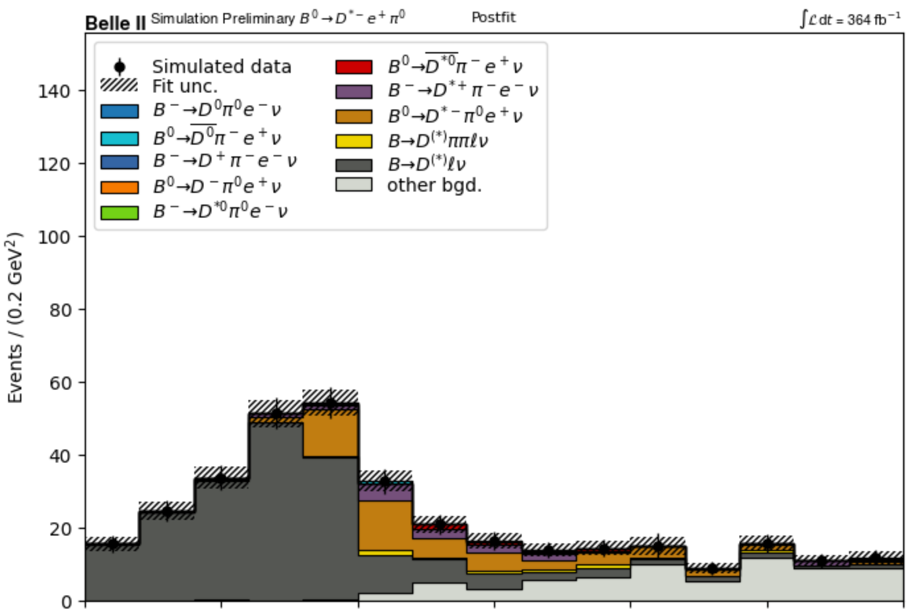
- Use scipy as initial guess

  → fit twice

  → fit converges quicker

```python
result = overall_fit.fit(
    observation=observation + model.config.auxdata,
    model=model,
    minimiser="scipy",
)
# get the scipy parameters
init_pars = result[0].tolist()
# do fit with minuit
result_minuit = overall_fit.fit(
    observation + model.config.auxdata,
    model,
    init_pars=init_pars,
    minimiser="minuit",
)
```

Results with Asimov Data

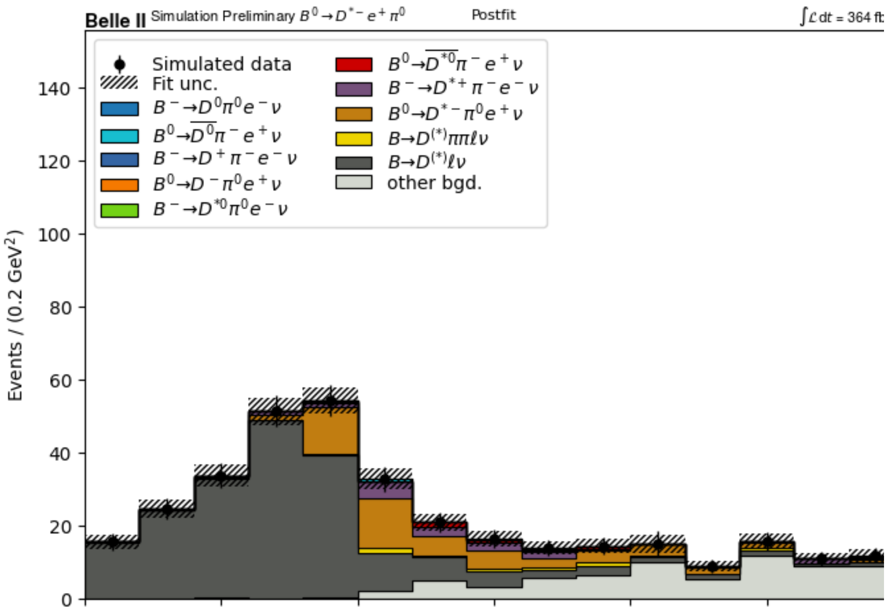



```python
def get_sample_order(
    model: pyhf.pdf.Model,
    mc_data: pd.DataFrame,
    component_column: str,
) -> List:
    """
    Function to get the ordering of the samples, so that can be filled into histogram afterwards directly.
    Better to automate this, in case I change the mc matching again.
    """
    sample_order_pyhf = model.config.samples
    sample_order_mc_matching = list(mc_data[component_column].cat.categories)
    sample_order = []
    for sample in sample_order_mc_matching:
        sample_order.append(sample_order_pyhf.index(sample))
    return sample_order
```

Hist module

- Sample ordering might have changed

Results with Asimov Data



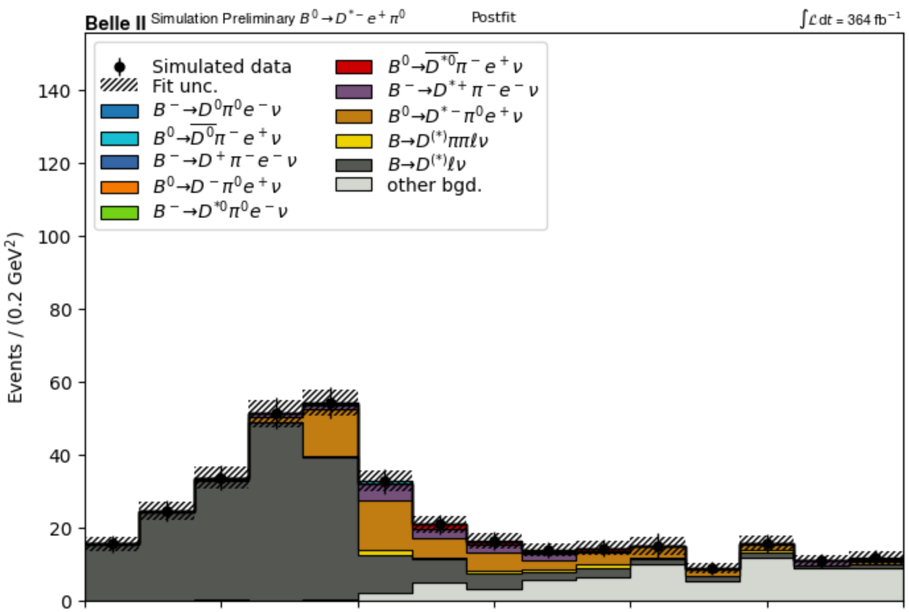

```python
def get_mc_counts(
    model: pyhf.pdf.Model,
    pars: np.ndarray,
    return_by_sample: bool = False,
) -> np.ndarray:
    pars = pyhf.tensorlib.astensor(np.asarray(pars))
    deltas, factors = model._modifications(pars)
    allsum = pyhf.tensorlib.concatenate(deltas + [model.nominal_rates])
    nom_plus_delta = pyhf.tensorlib.sum(allsum, axis=0)
    nom_plus_delta = pyhf.tensorlib.reshape(
        nom_plus_delta, (1,) + pyhf.tensorlib.shape(nom_plus_delta)
    )
    allfac = pyhf.tensorlib.concatenate(factors + [nom_plus_delta])
    newbysample = pyhf.tensorlib.product(allfac, axis=0)
    if return_by_sample:
        batch_first = pyhf.tensorlib.einsum("ij...->ji...", newbysample)
        if model.batch_size is None:
            return batch_first[0]
        return batch_first

    newresults = pyhf.tensorlib.sum(newbysample, axis=0)
    if model.batch_size is None:
        return newresults[0]
    return newresults
```

- get results after the fits

  → interpolation done automatically for nuisance parameters

  → returns values per bin

Get Errors using cabinetry




```
(
    total_stdev_model_bins,
    total_stdev_model_channels,
) = cabinetry.model_utils.yield_stdev(
    model, model_result[0][:, 0], model_result[0][:, 1], model_result[1]["corr"]
)
```

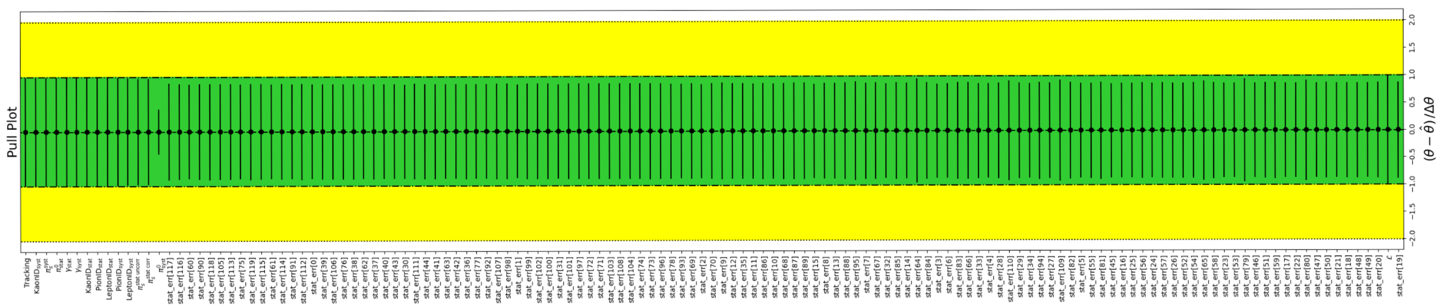
Fit value — Fit uncertainty — Correlation matrix

- for fit uncertainties:
- subtract statistical error from fit uncertainties in quadrature

  → one fit with scaling parameters only (statistical uncertainty)

  → then fit with everything

Pull distributions



- investigate pull distributions
  → code taken from this example
- possible to change label names

```python
pulls = pyhf.tensorlib.concatenate(
    [
        (
            bestfit[model.config.par_slice(k)]
            - model.config.param_set(k).suggested_init
        )
        / model.config.param_set(k).width()
        for k in model.config.par_order
        if model.config.param_set(k).constrained
    ]
)
# calculate the pull errors
pullerr = pyhf.tensorlib.concatenate(
    [
        errors[model.config.par_slice(k)] / model.config.param_set(k).width()
        for k in model.config.par_order
        if model.config.param_set(k).constrained
    ]
)
# get the labels for the pull plots
labels = np.asarray(
    [
        f"{k}[{i}]" if model.config.param_set(k).n_parameters > 1 else k
        for k in model.config.par_order
        if model.config.param_set(k).constrained
        for i in range(model.config.param_set(k).n_parameters)
    ]
)
```

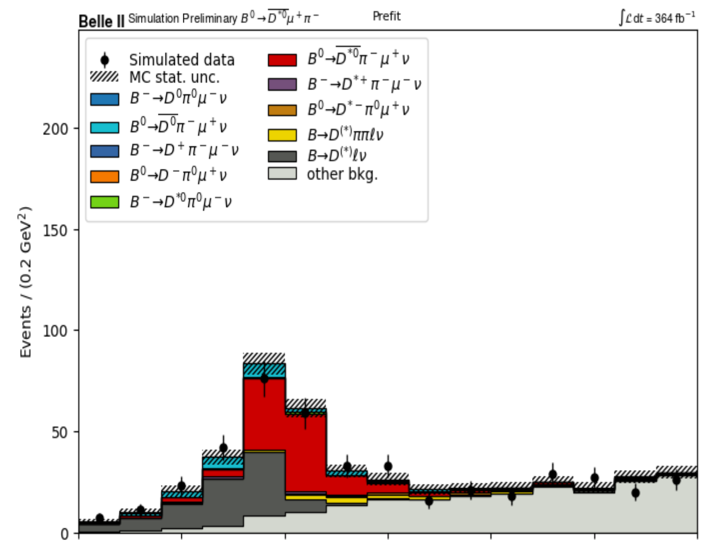
GEORG-AUGUST-UNIVERSITÄT GÖTTINGEN IN PUBLICA COMMODA SEIT 1737

pyhf ∂ifferentiable ℒikelihoods

Belle II

Toy Study

```python
def generate_toys(
    model: pyhf.pdf.Model,
    random_seed: int,
    n_toys: int = 500,
) -> np.ndarray:
    """
    Return a Tuple with a n_toys performed for a certain model.
    """
    np.random.seed(random_seed)
    toys = model.make_pdf(
        pyhf.tensorlib.astensor(np.asarray(model.config.suggested_init()))
    ).sample((n_toys,))
    return toys
```



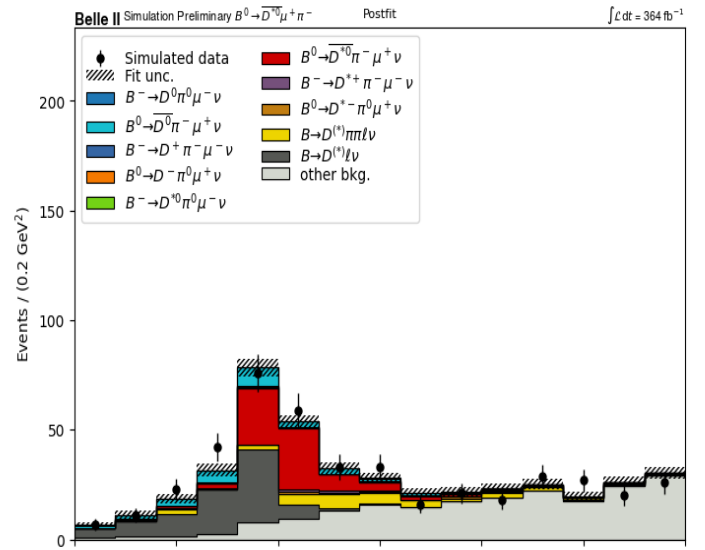
- Use histogram and Poisson smear every bin

  → looks more data like

- Histogram before the fit

Toy Study

```python
def generate_toys(
    model: pyhf.pdf.Model,
    random_seed: int,
    n_toys: int = 500,
) -> np.ndarray:
    """
    Return a Tuple with a n_toys performed for a certain model.
    """
    np.random.seed(random_seed)
    toys = model.make_pdf(
        pyhf.tensorlib.astensor(np.asarray(model.config.suggested_init()))
    ).sample((n_toys,))
    return toys
```



- Use histogram and Poisson smear every bin
  → looks more data like

- Histogram after the fit

Systematic Uncertainties

- possible to build workspace and remove particular modifier

  → use to determine effect of systematic uncertainties

```
new_workspace = workspace.prune(
    modifiers=not_considered_modifiers
)
```

- fit with all systematic uncertainties
- remove particular systematic uncertainty to determine impact on overall result

  → also automated that

```
syst_evaluation = [        You, 4 months ago • Included some changes from Mi
    "all",
    "statistics",
    "stat_err",
    "pi0",
    "lumi",
    "LeptonID",
    "PionID",
    "KaonID",
    "slow_pion",
    "slow_gamma",
    "Tracking",
]
# loop throw the list with the considered systematic uncertainties
for exclude in syst_evaluation:
    if exclude != "all":
        if exclude == "statistics":
            not_considered_modifiers = [
                modifier
                for modifier in model.config.par_order
                if "mu" not in modifier
            ]
            new_workspace = workspace.prune(
                modifiers=not_considered_modifiers
            )
        else:
            not_considered_modifiers = [
                modifier
                for modifier in model.config.par_order
                if exclude in modifier
            ]
            new_workspace = workspace.prune(
                modifiers=not_considered_modifiers
```

Conclusion

- PyHF works well to determine branching ratio of $B \to D^{**}\ell\nu$
- examples good to get a first understanding

  → more complex example for binned HEP analysis without hypotesting would be helpful
- need to determine bin counts and uncertainties yourself

  → has been automated for this analysis