# MC Reweighting Tools and Unfolding

Open LHCb Workshop on Semileptonic Exclusive $b \to c$ Decays

Markus Prim – 13/04/2023

# EFFORT

https://github.com/MarkusPrim/eFFORT2

# MC Reweighting Tools - EFFORT

**Reweighting & Fitting**

- Exclusive semileptonic rates:
  - $B \to D^{(*)}\ell\nu$
  - $B \to \pi\ell\nu, B \to \rho\ell\nu,$ $B \to \omega\ell\nu, (B \to \eta^{(\prime)}\ell\nu)$
- Modelling of inclusive $b \to u\ell\nu$ decays:
  - Hybrid MC approach
- Focus on SM and light leptons $\ell = e, \mu$

**Current Status**

- Current rework of the original version includes
  - Speedup (for fitting)
  - Improved design for modularity
  - Support for the uncertainty package
- Not everything supported in version 2 yet (implementation on demand)

# EFFORT – An Overview

Implementation has to provide the defined interface (helicity amplitudes)

```
# Define the BGL form factors with central values
BzeroToDStarBGL = BToDStarBGL(
    m_B=m_Bzero,
    m_V=m_Dstarplus,
    exp_coeff_a = param_central_values_MC[0:3],
    exp_coeff_b = param_central_values_MC[3:6],
    exp_coeff_c = param_central_values_MC[6:9],
)
```

BtoV or BtoP implements the Lorentz structure of the decay, form factors provide the helicity amplitudes

```
# Option A:
# Use rate with the form factor / helicity amplitudes
rate_B0 = BtoV(BzeroToDStarBGL, Vcb=vcb, eta_EW=1.0066)
```

```
# Option B:
# Use rate with the angular coefficients
angular_B0 = AngularCoefficients(BzeroToDStarBGL)
rate_B0Angular = BtoVAngular(angular_B0, Vcb=vcb_cln)
```

Available form factors are:
$b \to c$: BGL / CLN / (BLPRXP)
$b \to u$: BCL / BSZ

Alternatively:
Calculate angular coefficients with given form factors, and use the rate defined through angular coefficients

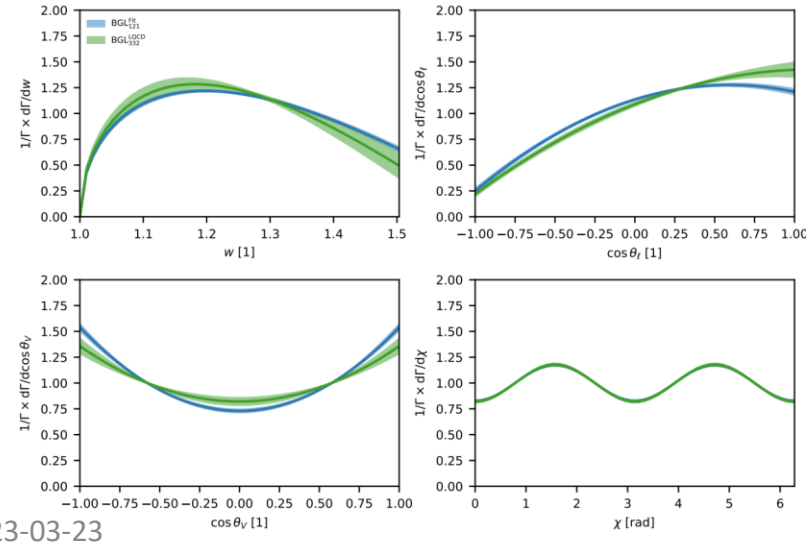
# EFFORT – An Overview

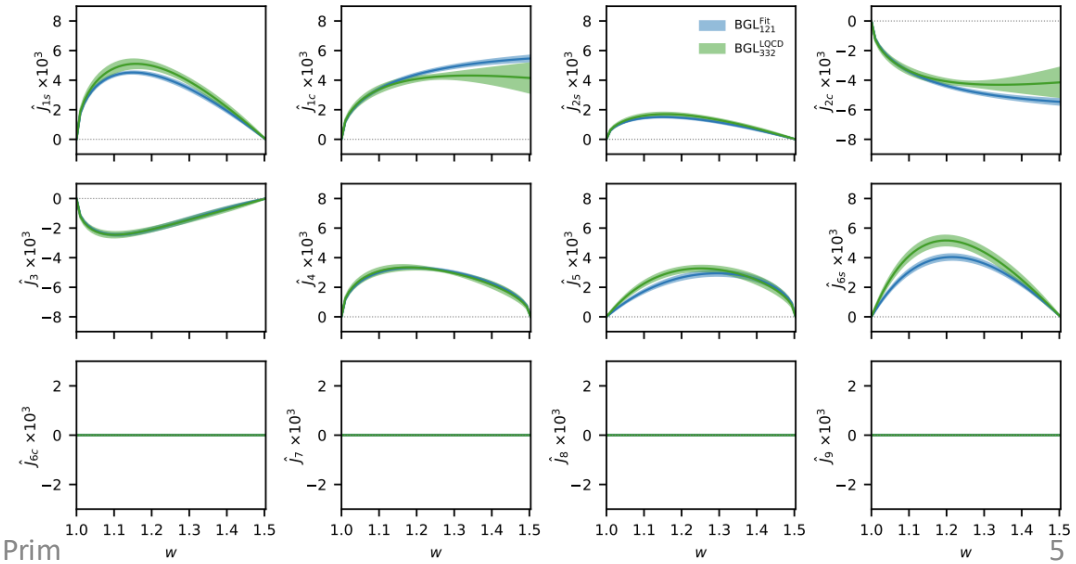
- Provides
  - 1D and 4D differential and partial rates
  - underlying form factors and angular coefficients

```
# Selected member functions (work for both rate classes)
# 4D differential rate
rate_B0.dGamma_dw_dcosL_dcosV_dchi(w, cosL, cosV, chi)
# 1D marginal rates
rate_B0.dGamma_dw(w)
rate_B0.dGamma_dcosL(cosL)
rate_B0.dGamma_dcosV(cosV)
rate_B0.dGamma_dchi(chi)

# 4D partial rates:
rate_B0.DGamma_Dw_DcosL_DcosV_Dchi(
    w1, w2, cosL1, cosL2, cosV1, cosV2, chi1, chi2)

# Access form factors, e.g.
rate_B0.FF.h_A1(w)
```



# EFFORT – Use Case 1: Reweighting

Requires:
MC Truth of w, cosL, cosV, chi

Weights between two parameterizations can be directly calculated

$$w = \frac{\Gamma_{old}}{\Gamma_{new}} \times \frac{d\Gamma_{\text{new}}}{d\Gamma_{old}}$$

```
# Define the BGL form factors with central values
BzeroToDStarBGL = BToDStarBGL(
    m_B=m_Bzero,
    m_V=m_Dstarplus,
    exp_coeff_a = param_central_values_MC[0:3],
    exp_coeff_b = param_central_values_MC[3:6],
    exp_coeff_c = param_central_values_MC[6:9],
)

BzeroToDStarBGL_new = BToDStarBGL(
    m_B=m_Bzero,
    m_V=m_Dstarplus,
    exp_coeff_a = param_central_values_new[0:3],
    exp_coeff_b = param_central_values_new[3:6],
    exp_coeff_c = param_central_values_new[6:9],
)
```

```
# Calculate weights
def weight(new, old, *x):
    tmp = old.Gamma() / new.Gamma()
    n = new.dGamma_dw_dcosL_dcosV_dchi(*x)
    o = old.dGamma_dw_dcosL_dcosV_dchi(*x)
    return tmp * n / o

# Calculate weight for a given phase space point
weight(BzeroToDStarBGL_new, BzeroToDStarBGL,
    w, cosL, cosV, chi)
```

# EFFORT – Use Case 2: Fitting

Setup required class (same as before)

Fitting update

Prediction

Chi2 to be used with the minimizer of your choice

```python
# Set up a form factor class for fitting
BzeroToDStarBGL_fit = BToDStarBGL(
    m_B=m_Bzero,
    m_V=m_Dstarplus,
    exp_coeff_a = np.zeros(3),
    exp_coeff_b = np.zeros(3),
    exp_coeff_c = np.zeros(3),
)

# Set up the angular coefficients
# (or the rate, depending on what you want to fit)
angular_B0_fit = AngularCoefficients(BzeroToDStarBGL_fit)
```

```python
def prediction(x):
    """Calculate prediction based on form factors x."""
    angular_B0_fit.FF.set_expansion_coefficients(x[:3], x[3:6], x[6:9])
    return np.array([
        *np.array([angular_B0_fit.DJ_Dw(
            angular_B0_fit.J1s, w_lower, w_upper) / angular_B0_fit.Norm()
            for w_lower, w_upper in w_bin_edges])     ,
        ...
    ])
```

```python
def chi2(a0, a1, a2, b0, b1, b2, c1, c2):
    """Calculate chi2 based on prediction from form factors
    and experimental data."""
    x = np.array([a0, a1, a2, b0, b1, b2, c1, c2])
    delta = (prediction(x) - experimental_data)
    chi2 = delta @ invC @ delta
    chi2 += (hA1_2014.n - angular_B0_fit.FF.h_A1(1))**2 / hA1_2014.s ** 2
    return chi2
```

# EFFORT – Summary

- Lightweight modular python package for reweighting and fitting of semileptonic exclusive decays
  - HAMMER is much more powerful but has a more complex interface overhead.
- Rather mature and used in several publications
- In-house tool, no “stable release” yet. It is around the corner, if you want to use it let me know and I will tag a stable release rather sooner than later

# PyRooUnfold

https://github.com/lucao-git/PyRooUnfold

https://gitlab.cern.ch/RooUnfold/RooUnfold

# (Py)RooUnfold – An Overview

**A question of the bias-variance tradeoff**

| Method | Description | Parameters | Comments | Function in PyRooUnfold |
|---|---|---|---|---|
| Bin-by-bin correction | unregularised, correct bin content with MC bin-by-bin factors | - | assumes no inter-bin migration; could have biases from the MC model | do_BinByBin |
| Matrix inversion | unregularised matrix inversion with singular value removal | - | give large bin-bin correlations and magnify statistical fluctuations; not accurate for small matrices | do_Invert |
| TUnfold | matrix inversion with 0-, 1-, or 2-order polynomial, an adjustable regularisation term of neighbouring bins | reg. strength $\tau$ | optimal tau can be chosen by scanning L-curve; better when nbin_mea > nbin_true | do_TUnfold($\tau$) |
| Bayes | use Bayes' theorem to invert response matrix, regularisation by stopping iterations before reaching "true" inverse | iteration n | n needs to be tuned with statistics, bins | do_Bayes( $n$ ) |
| IDS | iterative, dynamically stabilised method, uses stat. significance of the data-MC differences in each bin for regularisation | iteration n | allows to treat the effects of new structures in data and the large fluctuations from background subtraction | do_Ids( $n$ ) |
| SVD | singular value decomposition, regularisation with a smooth cut-off on small singular value contribution | $k$ = 1..nbins | k too small: dominated by MC truth<br>k too large: dominated by stat. fluctuations<br>k needs to be tuned with bins, sample size | do_Svd( $k$ ) |

L. Cao

# (Py)RooUnfold – An Overview

**RooUnfold**

- Mature, maintained, and documented package based on ROOT
- Provides the unfolding methods, but no evaluation of the methods/parameters

**PyRooUnfold**

- Not only a python wrapper around RooUnfold
- Provides automated evaluation of methods and parameters

# PyRooUnfold - Capabilities

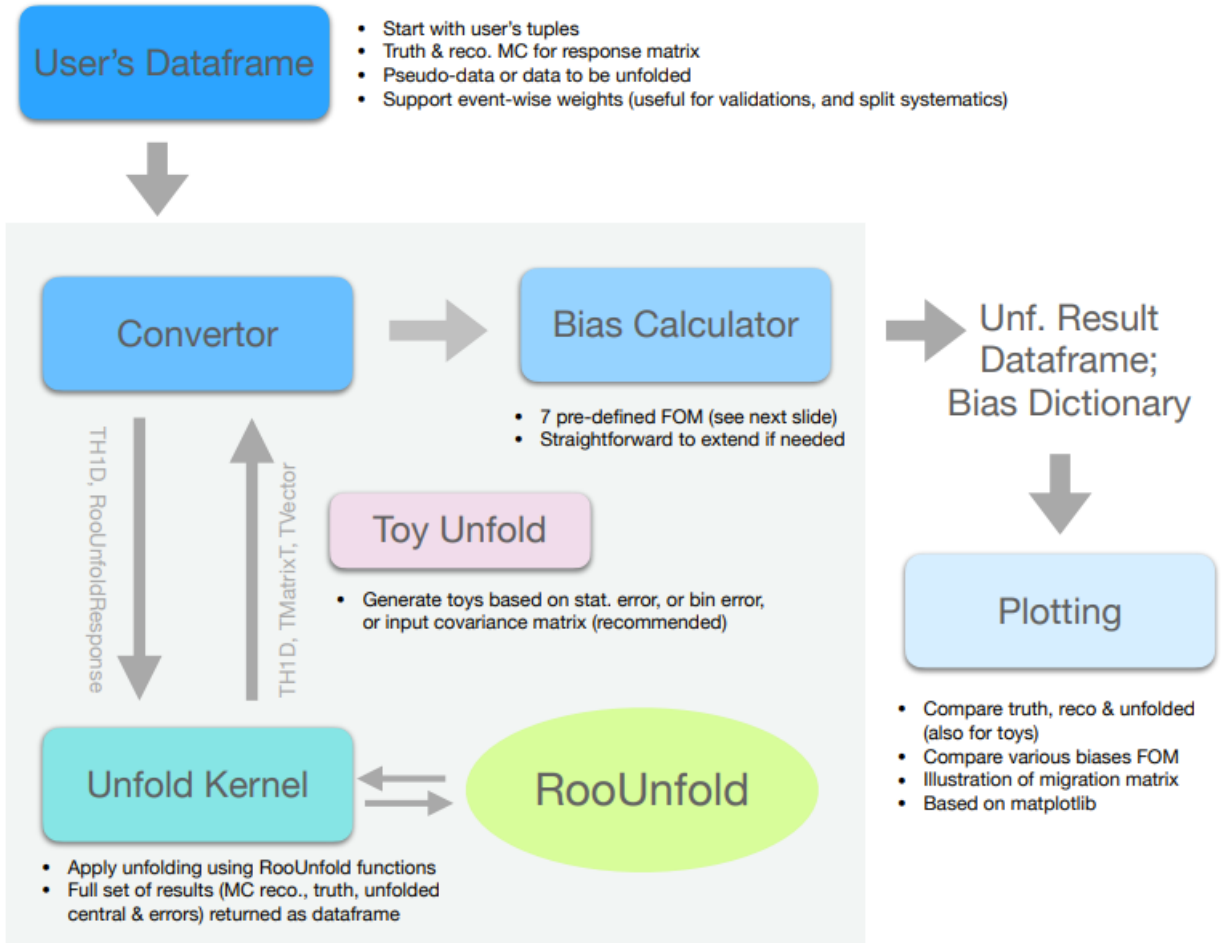
Not only **A Python wrapper of RooUnfold**

**Included in the current version:**

- Interface of all above methods and major features of RooUnfold
- Toys generation with input error/cov.
- Bias calculator
- Commonly needed plotting functions

**Missing:**

- Support of non-square response matrix (different binning used for reco. and truth)
- Interface for multidimensional unfolding




L. Cao

# PyRooUnfold – Method Evaluations F.o.M

**Each valid but limited, proper checks require all**

| Figure of Merit | Description | Nickname |
|---|---|---|
| $\sum_i \lvert b_i \rvert$ | Sum of absolution biases in each bin, $b_i = N_{\text{unfolded}} - N_{\text{true}}$ | a |
| $\sum_i \lvert b_i \rvert / N_i^{\text{true}}$ | Sum of normalized absolute biases $(N_i^{\text{true}} \neq 0)$ | b |
| $\sum_i b_i$ | Sum of biases, which can be zero when significant biases are present in individual bins but cancel in the sum (balanced oscillation) | c |
| $\sqrt{\sum_{i,j} Cov_{i,j}}$ | The square root of the sum of all elements of the post-unfold covariance matrix. | d |
| $\sum_i \frac{\lvert b_i \rvert}{\sqrt{\sum_{i,j} Cov_{i,j}}}$ | Ratio of the sum of absolute biases and the error taking into account bin-to-bin correlations. | e |
| $\sqrt{\left(\sum_i \lvert b_i \rvert\right)^2 + \sum_{i,j} Cov_{i,j}}$ | The total biases absorbing the total error with bin-to-bin correlations, which is used to check the general oscillation. | f |
| $\sum_i \frac{\lvert b_i \rvert}{\sqrt{Cov_{ii}}}$ | The summed ratio of the absolute bias and unfolding error in each bin. | g |

L. Cao

# PyRooUnfold – An Example

```
from pyroounfold.unfold import unfold

example_q2 = unfold(

    df_train = df_mc,
    weight_train = df_mc['weight_train'],

    df_test = df_mc,
    weight_test = df_mc['weight_test'],

    name_var_true = 'true_q2',
    name_var_reco = 'reco_q2',

    show_var = r'$q^{2}$',# for plotting

    bins = q2_bins

    #optional input:

    reco_bin_error

    reco_cov

    kcovtoy

    mc_stat_err

    )
```

← df for defining migration matrix  Asimov unfolding, if train = test

← df of unfolding target, can be re-set by example_q2.**set_hist_measure**(bin_centre, bin_error)

← column names of variables

← binning of variables

← reco. bin-wise uncertainties need to propagated in unfolding

← reco. covariance matrix, alternative of bin-wise uncertainties

← error propagation method (RooUnfold): False - based on input reco_cov, True - based on internal toys

← effect of MC statistics (RooUnfold): 0 - exclude, 1 - include, 2 - only

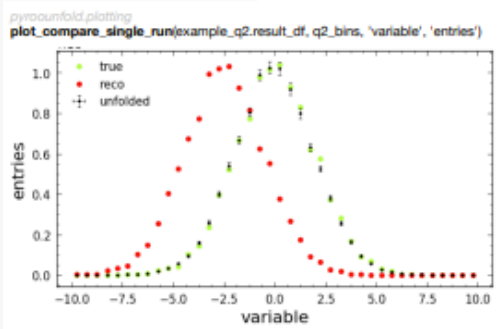
L. Cao

# (Py)RooUnfold – An Overview

```
example_q2.do_Invert()
```
← apply unfolding using matrix-inversion method

```
example_q2.result_df
```
← result of unfolding

| | bin_index | truth_central | truth_stat_error | measured_central | measured_error | unfolded_central | unfolded_error |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0.0 | 0.000000 | 3.0 | 1.732051 | 0.557234 | 3.488718 |
| 1 | 1 | 0.0 | 0.000000 | 5.0 | 2.236068 | 0.681334 | 3.522817 |
| 2 | 2 | 0.0 | 0.000000 | 6.0 | 2.449490 | 1.011481 | 3.699059 |
| 3 | 3 | 1.0 | 1.000000 | 22.0 | 4.690416 | 1.271691 | 2.895884 |
| 4 | 4 | 0.0 | 0.000000 | 34.0 | 5.830952 | 2.223960 | 2.711121 |
| 5 | 5 | 3.0 | 1.732051 | 47.0 | 6.855655 | 3.003067 | 1.540172 |
| 6 | 6 | 4.0 | 2.000000 | 102.0 | 10.099505 | 4.560288 | 1.656089 |



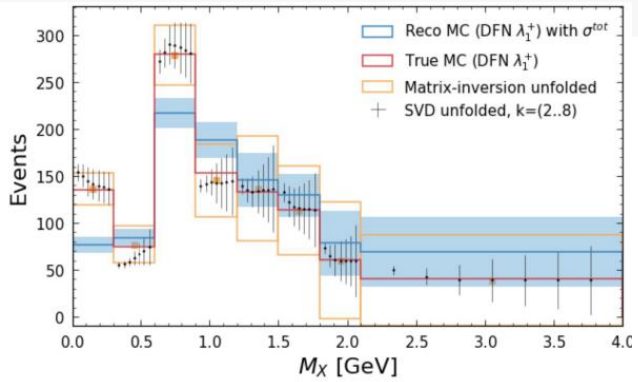

```
example_q2.result_cov
```
← post-unfolding covariance matrix

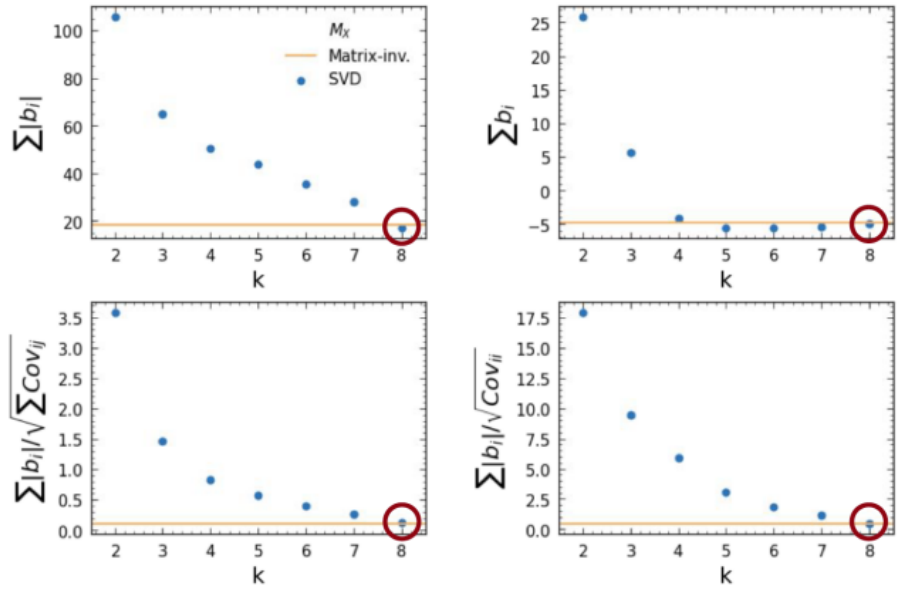
Defines k cutoff

```
example_q2.do_Svd(5)
```

```
example_q2.check_bias()
```
← calculate and save bias F.O.M.

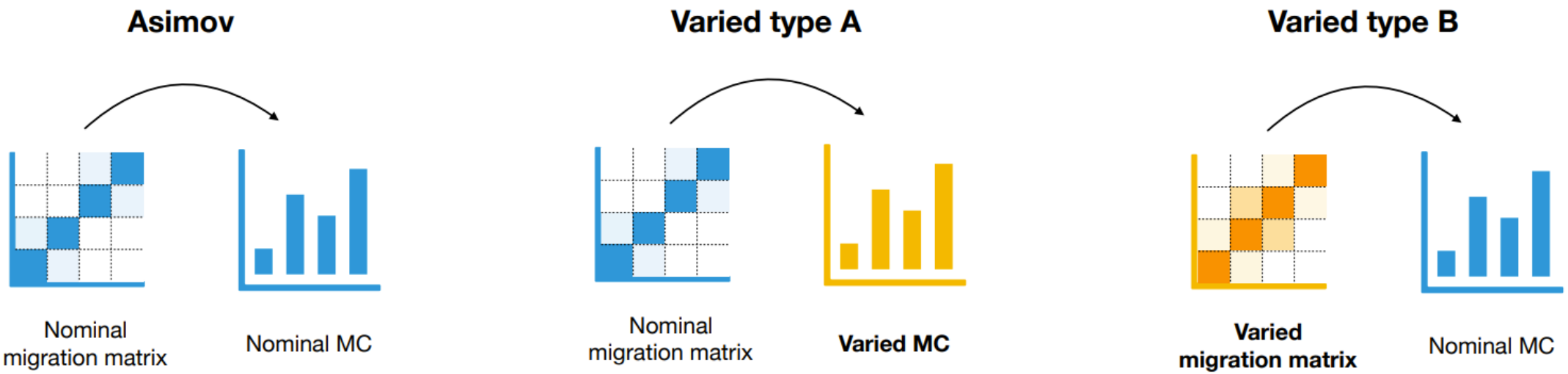
```
example_q2.bias_a
example_q2.bias_b
```
← results of bias F.O.M, bias_a, …, bias_g



# How to Treat Systematic Effects

Effects of e.g., form factor uncertainties on the unfolding uncertainty




Reweight MC Truth using e.g., EFFORT and repeat unfolding and acceptance correction **simultaneously**

# Summary

- This was NOT an exhaustive list of available tools
- EFFORT and PyRooUnfold
  - Lightweight python frameworks to streamline reoccurring tasks
  - Both tools have been used in Belle (II) publications
  - If you miss a feature, contact the developers (mail / issue)