Hands-on session on **basf2**

Harsh Purwar

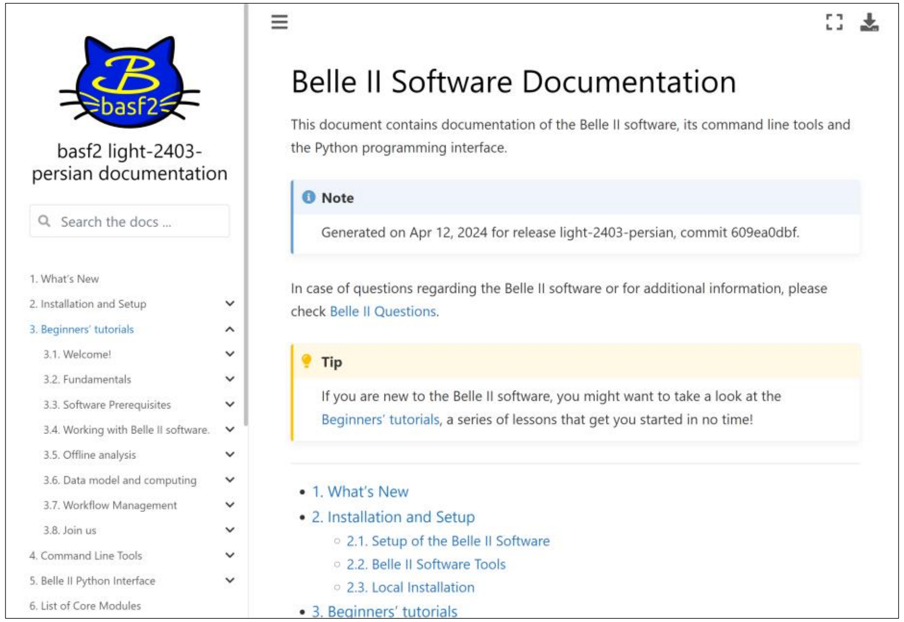
Postdoctoral Researcher
HEP Group, Dept of Phys & Astr,
University of Hawaii at Manoa (UHM), Honolulu, HI, USA
Email: purwar@hawaii.edu

Belle II Summer Workshop – 2024
Hosted by the Univ. of Mississippi, Oxford, MI

basf2 – Introduction

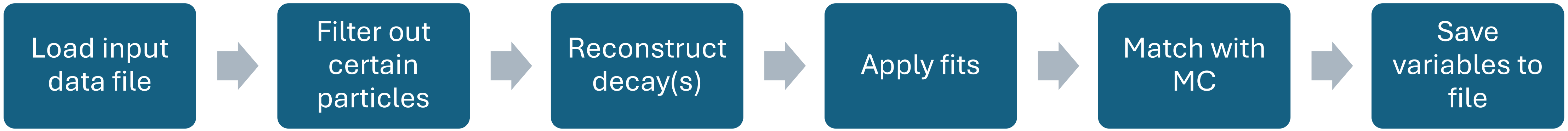
- **basf2** : Belle II Analysis Software Framework
- Essentially, it provides a set of tools (variables, methods, modules, etc.) that make analyzing the Belle II particle collisions data easier – apart from several other things!
- Documentation: https://software.belle2.org
    - The link also lists various recent releases of the software framework.

Sphinx documentation	Doxygen documentation
light-2403-persian (recommended)	light-2403-persian
light-2311-nebelung	light-2311-nebelung
light-2309-munchkin	light-2309-munchkin
light-2305-korat	light-2305-korat
release-08-01-05 (recommended)	release-08-01-05
release-08-00-08	release-08-00-08
release-06-02-00	release-06-02-00
release-06-01-15	release-06-01-15
release-06-00-14	release-06-00-14
release-05-02-19	release-05-02-19
release-05-01-25	release-05-01-25
development	development



`basf2` – Introduction

- The software framework provides you with a python interface for calling C++ modules or functions that operate on data objects
- Order/sequence of these operations on data is defined by a `Path` variable



- In our python (steering) file, we essentially define this `Path`

basf2 Releases

1. **Main releases** (release-major-minor-patch, eg: `release-08-01-06`)
   - Includes all approved changes and suitable for everything, data-taking, analysis, MC data production, etc.
2. **Light releases** (light-yymm-cat breed, eg: light-2405-quaxo)
   - Includes only libraries necessary for analysis, updated with most bugfixes and features
   - Designed for high-level analysis only (can't process anything other than mdst/udst files)

Prerequisites

1. Active KEK computing account
   - able to ssh to kekcc
2. Active DESY account
   - able to clone from gitlab.desy.de on kekcc
3. Basic knowledge of Linux, git, python, Jupyter notebook, matplotlib, NumPy, pandas

Running `basf2`

- Run your steering file with:

```
basf2 [OPTIONS] [STEERING_FILE] [-- [STEERING_FILE_OPTIONS]]
```

Some useful command-line options `[OPTIONS]`:

- NOTE: THESE OPTIONS HAVE HIGHER PRIORITY THAN THOSE IN YOUR STEERING FILE
  - `--dry-run` : Useful for checking errors, etc., it doesn't actually execute the Path over various events
  - `-n <N>` : Limits execution to **N** events.
  - `-i` : Specify the input filename.
  - `-o` : Specify the output filename.
  - `--help` : Prints full list of available command-line options.

NOTE:
If your steering file takes a significant amount of time (> few hours) to execute, please submit it as a job on kekcc using `bsub` (usage: https://kekcc.kek.jp/service/kekcc/support/en/04/)

Analysis we'll work on...

- This year we are going to try out a missing particle (ν) analysis
- Decay chain/mode to reconstruct:

$$\overline{B^0} \rightarrow D^{*+} + e^- + \overline{\nu}_{e^-}$$
$$D^{*+} \rightarrow D^0 + \pi^+$$
$$D^0 \rightarrow K^- + \pi^+$$

- Awesome! But what data do we use? Belle II collisions data? **– Nope not so fast!**
- **You start with MC data.**
  - Ideally, use already generated MC data made available to us by the Data Production Group.
  - In certain scenarios though, you may need to generate your own MC data for the analysis to get started.
- For this session, we'll use MC data file on kekcc: `/group/belle2/users2022/purwar/b2sw2024/B2Dsenu_SM_100k.root`

Getting started with **basf2**

1. Setup a light release of **basf2** from CVMFS on kekcc:
   `source /cvmfs/belle.cern.ch/tools/b2setup light-2403-persian`
   - *Hack:* Add it to your ~/.bashrc

```
source /cvmfs/belle.cern.ch/tools/b2setup light-2403-persian
```

   - *Better:* Create an alias in your ~/.bashrc

```
alias bs2Light='source /cvmfs/belle.cern.ch/tools/b2setup light-2403-persian'
```

```
[purwar@ccw05 ~]$ bs2Light
Belle II software tools set up at: /cvmfs/belle.cern.ch/tools
Environment setup for release: light-2403-persian
Central release directory    : /cvmfs/belle.cern.ch/el7/releases/light-2403-persian
[purwar@ccw05 ~]$

[purwar@ccw05 ~]$ basf2 --info
```

Hands-on session – Example scripts

Available on DESY Gitlab:

```
git clone git@gitlab.desy.de:purwar/b2sw2024.git
```

Feel free to have a look at the example scripts!

Structure of a basic steering file for reconstruction

- import basf2, modularAnalysis, other packages

- Start with a path, main = basf2.Path()

- Specify input MDST file: inputMdst, inputMdstList

- Load/fill particle lists: fillParticleList, fillParticleLists
- Apply any cuts, if needed: applyCuts

- If electrons are in the final state particle list, consider Bremsstrahlung correction: correctBrems

- Build Event Kinematics, if needed: buildEventKinematics

- Start reconstructing your decay chain: reconstructDecay
- Apply cuts, if needed: applyCuts

- Match MC Truth: matchMCTruth

- Apply vertex fit: K-fit or tree fit, if needed: vertex.treefit
- If multiple reconstructed particles, apply best candidate selection (BCS): rankByHighest

- Build Rest of Event, append ROE mask, apply any necessary cuts: buildRestOfEvent, appendROEMask

- Dump all variables to be exported in a list
- Write out these variables as Ntuples to the output root file: VariablesToNtuple

- Execute path: basf2.process(main)

Decay string syntax

Commonly used with reconstructDecay() and for creating aliases

- “Mother particle” arrow “daughter particle(s)”: `D0:kpi -> K-:loose pi+:loose`
- To construct a decay sequence, use square brackets: `D*+ -> [D0 -> K- pi+] pi+:slow`

Arrows in decay strings		
Arrow types	**Intermediate resonances**	**Radiated photons**
`->` (default)	✓	✓
`=direct=>`	✗	✓
`=norad=>`	✓	✗
`=exact=>`	✗	✗

Note: Different arrows are allowed in same decay str, `D*+ -> [D0 =direct=> K- pi+ pi0] pi+`

Full documentation: https://software.belle2.org/light-2403-persian/sphinx/analysis/doc/DecayString.html

Decay string syntax

Commonly used with reconstructDecay() and for creating aliases

What would be the full decay string for our decay?

$$\overline{B^0} \rightarrow D^{*+} + e^- + \overline{\nu}_{e^-}$$
$$D^{*+} \rightarrow D^0 + \pi^+$$
$$D^0 \rightarrow K^- + \pi^+$$

- How do we specify the missing electron-neutrino?

Full documentation: https://software.belle2.org/light-2403-persian/sphinx/analysis/doc/DecayString.html

Decay string syntax

Commonly used with reconstructDecay() and for creating aliases

What would be the full decay string for our decay?

$$\overline{B^0} \rightarrow D^{*+} + e^- + \overline{\nu}_{e^-}$$
$$D^{*+} \rightarrow D^0 + \pi^+$$
$$D^0 \rightarrow K^- + \pi^+$$

- How do we specify the missing electron-neutrino?

Including missing particles (`Keywords`):

- `'...'` Missing massive final state particles are ignored
- `'?nu'` Missing neutrinos are ignored
- `'?gamma'` Missing gammas are ignored
- `'?addbrems'` Gammas added by bremsstrahlung tools are ignored

Decay string syntax

Commonly used with reconstructDecay() and for creating aliases

What would be the full decay string for our decay?

$$\overline{B^0} \rightarrow D^{*+} + e^- + \overline{\nu}_{e^-}$$
$$D^{*+} \rightarrow D^0 + \pi^+$$
$$D^0 \rightarrow K^- + \pi^+$$

```
anti-B0:rec -> [D*+:d0p -> [D0:kp -> K-:sl pi+:sl] pi+:slow] e-:sl ?nu
```

Markers in decay string:

- ^ selection of succeeding particle
- @ succeeding particle is unspecified (inclusive decays)
- (misID), (decay), etc...

Full documentation: https://software.belle2.org/light-2403-persian/sphinx/analysis/doc/DecayString.html

Creating Aliases

```
import variables.variables as va
va.printAliases()
```

1

```
import variables.variables as va
# va.addAlias(‘alias’, ‘variable’)
```

2

```
import variables.utils as vu
# vu.create_aliases(listOfVariables, wrapper, prefix)
cmsVars = vu.create_aliases([‘E’,‘px’,‘py’,‘pz’],
                            'useCMSFrame({variable})', 'CMS')
```

3

```
import variables.utils as vu
# vu.create_aliases_for_selected(listOfVariables, decayStr, prefix)
vars = vu.create_aliases_for_selected([‘M’], ‘D0->^K- ^pi+’, [‘k’,‘pi’])
```

Missing particle analysis

- Import all necessary modules
- Append analysis global tag -- needed for MVA weights (used later)
- Create a path variable
- Create a list for output variables
- Input and output root filenames
- Import the MDST file

Any other required initializations

```
import basf2 as b2
import modularAnalysis as ma
import variables.utils as vu
from variables import variables as va
import vertex as vx

analysis_gt = ma.getAnalysisGlobaltag()
b2.conditions.append_globaltag(analysis_gt)

main = b2.Path()
outVars = []

inRootFile =
"/group/belle2/users2022/purwar/b2sw2024/B2Dsenu_SM_100k.ro
ot"
outRootFile = "./b2dsenu_SM_100k.root"

ma.inputMdst(environmentType='default',
filename=inRootFile, path=main)

noCuts = False
```

Missing particle analysis

```
va.addAlias("goodTrack", "passesCut(dr<2 and abs(dz)<4 and E<5.5 and thetaInCDCAcceptance and nCDCHits>5)")
va.addAlias("goodTrack_slow", "passesCut(dr<2 and abs(dz)<4 and E<5.5 and thetaInCDCAcceptance and nCDCHits>0)")

ma.fillParticleList("K-:sl", cut='goodTrack and pt>0.1 and kaonID>0.1', path=main)
ma.fillParticleList("pi+:sl", cut='goodTrack and pt>0.25 and pionID>0.1', path=main)
ma.fillParticleList("pi+:slow", cut='goodTrack_slow', path=main)
ma.fillParticleList('e-:sl', cut='dr<2 and abs(dz)<4', path=main)

ma.applyChargedPidMVA(['e-:sl'], path=main, trainingMode=1, chargeIndependent=False, binaryHypoPDGCodes=(0, 0))

ma.applyCuts('e-:sl',cut='thetaInCDCAcceptance and nCDCHits>5 and pidChargedBDTScore(11, ALL)>0.9 and p>0.2', path=main)
```

- Add few aliases for initial (pre-selection) cuts, like goodTrack, etc.
- Fill particle lists with all the final state particles
  - Use recommended cuts + some loose PID cuts
  - No CDC hits required for the slow pion
- You may add a condition using noCuts variable to visualize these cuts
- You may also add Bremsstrahlung correction for electrons

Missing particle analysis

```
ma.buildEventKinematics(fillWithMostLikely=True, path=main)

ma.reconstructDecay("D0:sl -> K-:sl pi+:sl", '', path=main)
ma.applyCuts('D0:sl', cut='abs(dM)<0.03', path=main)
ma.reconstructDecay("D*+:sl -> D0:sl pi+:slow", '', path=main)
ma.applyCuts('D*+:sl', cut='0.0025<Q<0.01', path=main)
ma.reconstructDecay("anti-B0:sl -> D*+:sl e-:sl ?nu", cut='', path=main)

ma.summaryOfLists(['K-:sl','pi+:sl','pi+:slow','e-:sl','D0:sl','D*+:sl','anti-B0:sl'], path=main)
```

- Build event kinematics with `fillWithMostLikely=True` – Needed for missing particle info

buildEventKinematics()

- Calculates the global kinematics of the event (visible energy, missing momentum, missing mass…) using:
  - ParticleLists provided, otherwise
  - default ParticleLists (all track and all hits in ECL without associated track)
- The visible energy missing values are stored in a EventKinematics dataobject.
- fillWithMostLikely – if True, the module uses the most likely particle mass hypothesis for charged particles according to the PID likelihood and the option inputListNames will be ignored.

```
buildEventKinematics
(
        inputListNames = None,
        default_cleanup = True,
        custom_cuts = None,
        chargedPIDPriors = None,
        fillWithMostLikely = False,
        path = None
)
```

Missing particle analysis

```
ma.buildEventKinematics(fillWithMostLikely=True, path=main)

ma.reconstructDecay("D0:sl -> K-:sl pi+:sl", '', path=main)
ma.applyCuts('D0:sl', cut='abs(dM)<0.03', path=main)
ma.reconstructDecay("D*+:sl -> D0:sl pi+:slow", '', path=main)
ma.applyCuts('D*+:sl', cut='0.0025<Q<0.01', path=main)
ma.reconstructDecay("anti-B0:sl -> D*+:sl e-:sl ?nu", cut='', path=main)

ma.summaryOfLists(['K-:sl','pi+:sl','pi+:slow','e-:sl','D0:sl','D*+:sl','anti-B0:sl'], path=main)
```

- Build event kinematics with `fillWithMostLikely=True` – Needed for missing particle info
- Reconstruct decays, apply cuts (again you may use noCuts variable to apply conditional cuts after visualizing them)
- summaryOfLists – Useful to see # of events you drop with cuts – output is a bit strange though!

Missing particle analysis

```
ma.matchMCTruth('anti-B0:sl', path=main)

vx.treeFit('anti-B0:sl', conf_level=0.001, ipConstraint=True, path=main)
ma.applyCuts('anti-B0:sl', cut='chiProb>0.05', path=main)

ma.rankByHighest('anti-B0:sl', variable='chiProb', numBest=1, path=main)

ma.summaryOfLists(['D0:sl','D*+:sl','anti-B0:sl'], path=main)
```

- MC matching for all particles (including daughters, granddaughters, etc.)
- Apply a vertex fitter (treeFit) with IP constraint – (more details)
- Apply a loose cut on chiProb = χ^2 probability of the fit
- Use chiProb for the Best candidate selection

Missing particle analysis

```
ma.buildRestOfEvent('anti-B0:sl', fillWithMostLikely=True, path=main)

trackBasedCuts = 'thetaInCDCAcceptance and pt > 0.1 and dr < 5 and abs(dz) < 10'
eclBasedCuts = 'thetaInCDCAcceptance and E > 0.05'
ma.appendROEMask('anti-B0:sl', 'myM', trackBasedCuts, eclBasedCuts, path=main)

ma.applyCuts('anti-B0:sl',cut='roeM(myM)<10 and roeMbc(myM)>4.25 and -4<roeDeltae(myM)<5 and -1.5<weMissM2(myM,3)<1.5',
path=main)
```

- Build rest of event (everything that not on the signal side is in ROE) (more details)
- Clean up the ROE using appendROEMask
- Apply more cuts to refine signal B

Missing particle analysis

```
listVars1 =
['M','InvM','dr','dz','dM','mcPDG','PDG','pt','E','p','px','py','pz','mcPX','mcPY','mcPZ','mcE','theta','nCDCHits']
listVars2 = listVars1 + ['isSignal','Q']
listVars3 = listVars2 + ['Mbc','deltaE','chiProb','mcErrors']

outVars += vu.create_aliases_for_selected(listVars1+['kaonID','pionID'], "anti-B0:sl -> [D*+:sl -> [D0:sl -> ^K-:sl
^pi+:sl] ^pi+:sl] e-:sl", prefix=['Km_sl','pip_sl','pip_slow'])
outVars += vu.create_aliases_for_selected(listVars2, "anti-B0:sl -> [^D*+:sl -> [^D0:sl -> K-:sl pi+:sl] pi+:sl] ^e-:sl",
prefix=['Dsp_sl','D0_sl','lm_sl'])

outVars += listVars3

roeKinematics = ['roeE(myM)', 'roeM()', 'roeM(myM)', 'roeP(myM)', 'roeMbc()', 'roeMbc(myM)',
                 'roeDeltae()', 'roeDeltae(myM)']
roeMultiplicities = ['nROE_Charged(myM)', 'nROE_Photons(myM)', 'nROE_NeutralHadrons(myM)']
weKinematics_3 = ['weMissE(myM,3)', 'weMissPx(myM,3)', 'weMissPy(myM,3)', 'weMissPz(myM,3)']
weMissM2_n = ['weMissM2(myM,3)']

outVars += roeKinematics + roeMultiplicities
outVars += weKinematics_3 + weMissM2_n
```

- Fill outVars with variables to be exported as ntuple for offline analysis/visualization

Missing particle analysis

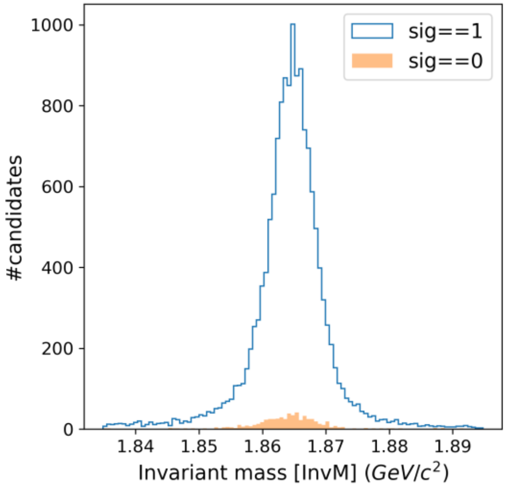
```
ma.variablesToNtuple('anti-B0:sl', variables=outVars, path=main, treename = 'antiB0', filename=outRootFile)

b2.process(main)
print(b2.statistics)
```

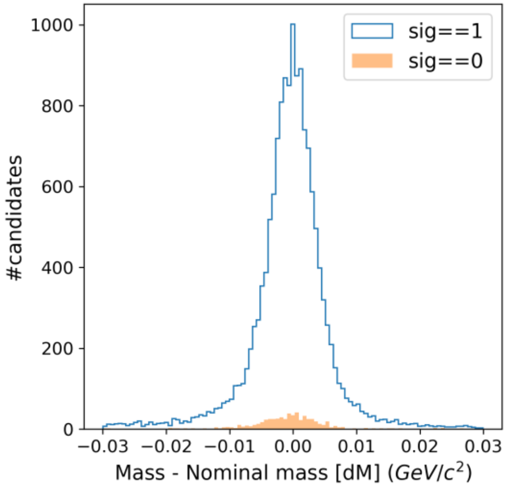
- Export the list of variables to ntuple

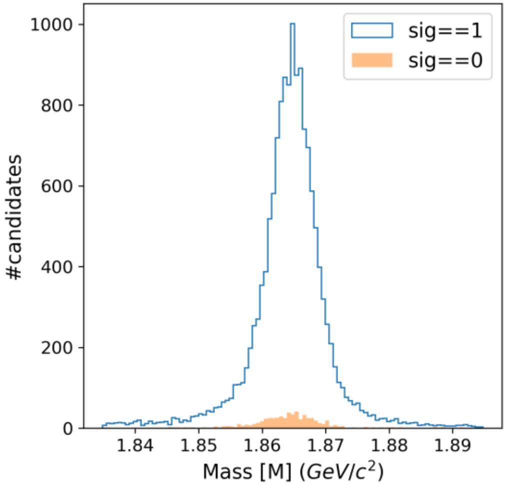
- Process the path
- Print some statistics (useful to debug if code is running very slow!)

Results – Reconstructed D^0 and D^{*+}

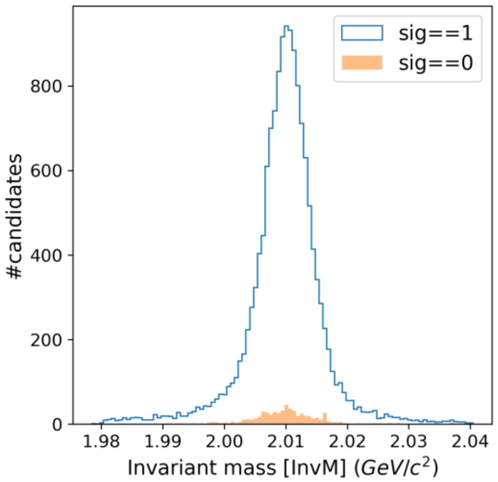
D^0

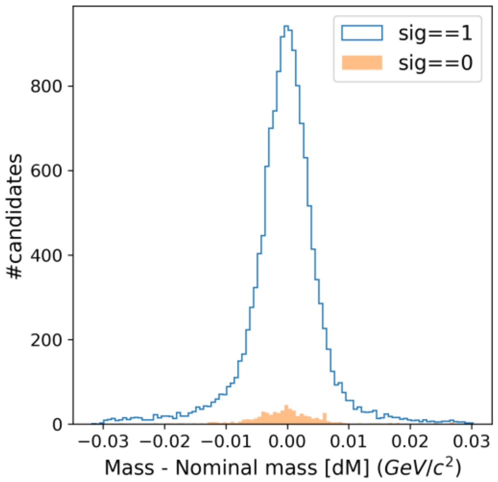


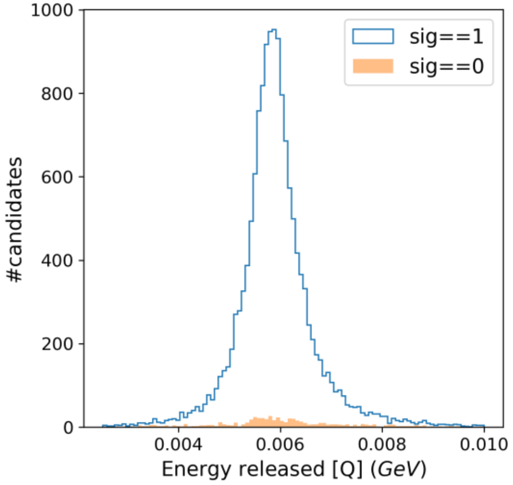




D^{*+}

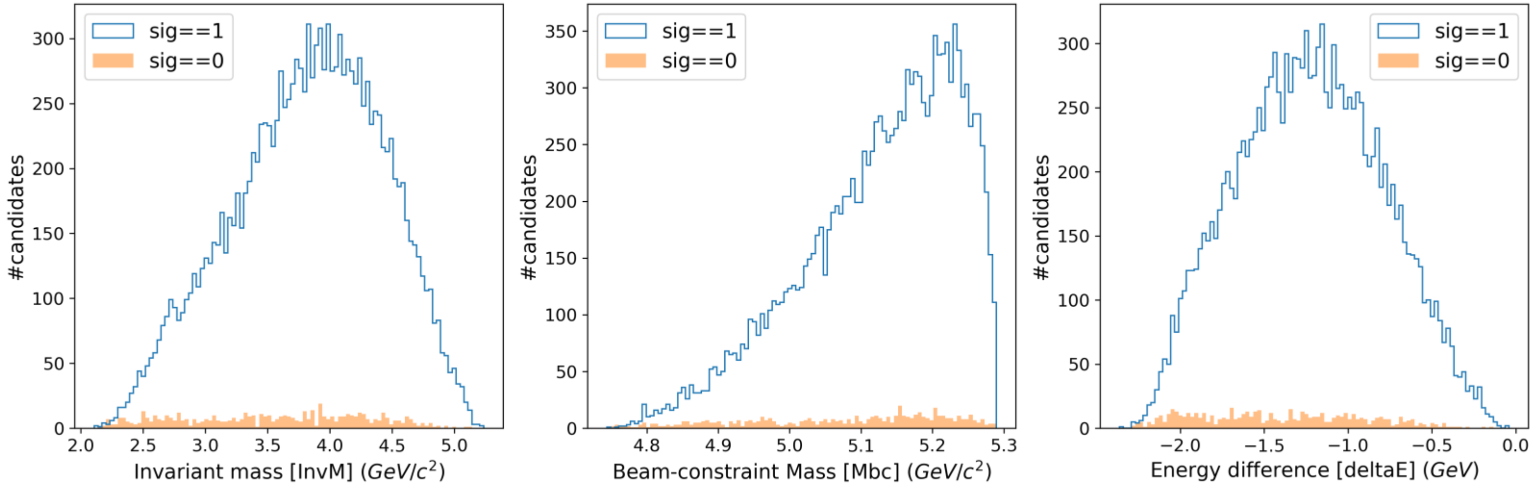




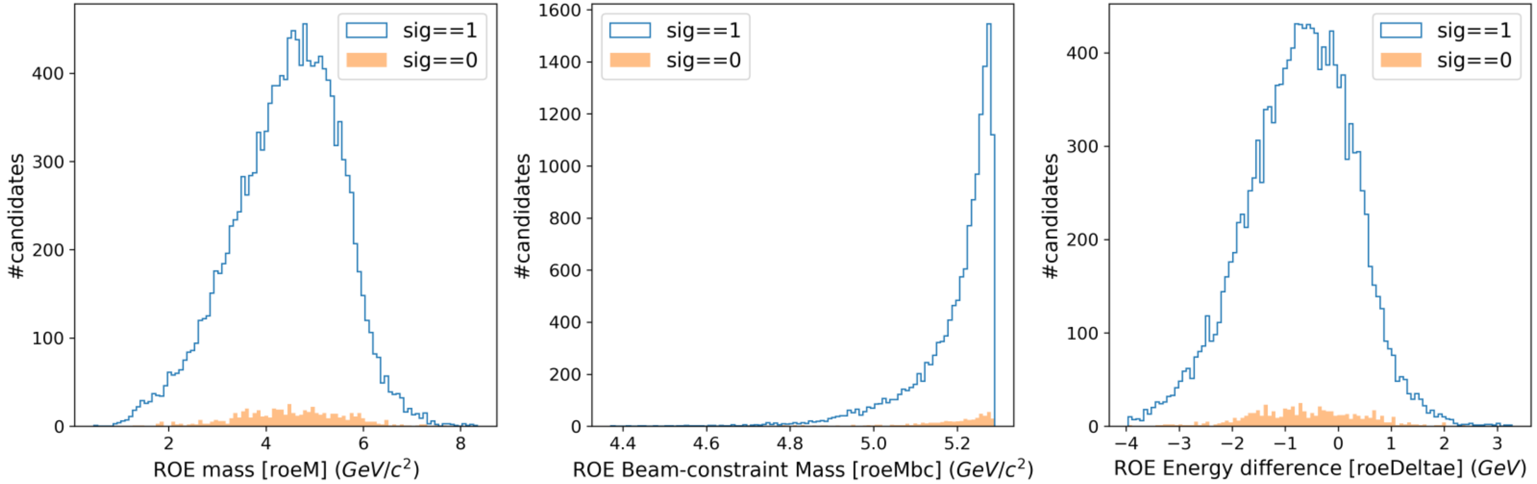


Results – Reconstructed $\overline{B^0}$ and ROE

$\overline{B^0}$



ROE



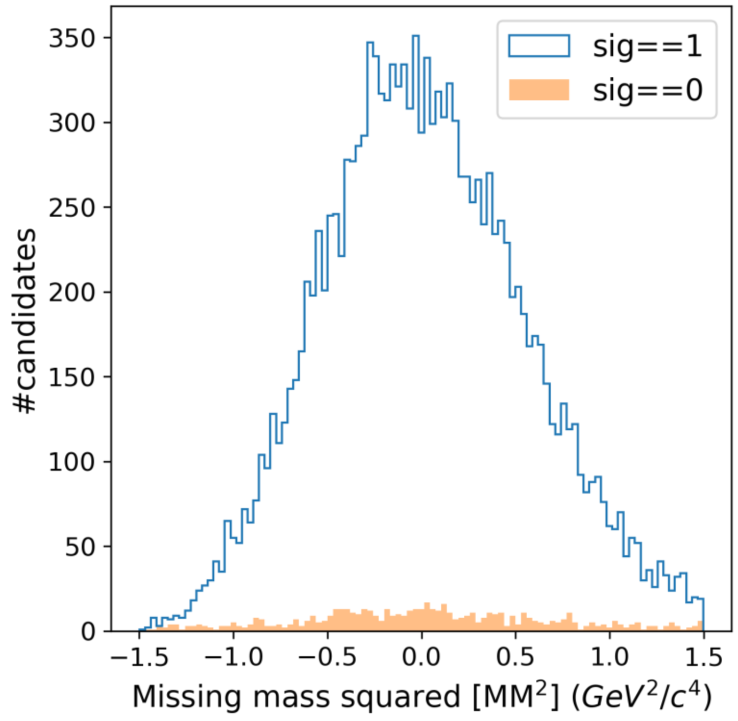
Missing mass squared (MM^2)

weMissM2(maskName, opt)

Returns the invariant mass squared of the missing momentum.

Possible options **opt** are the following:

- 0: CMS, use energy and momentum of charged particles and photons
- 1: CMS, same as 0, fix Emiss=$p$miss
- 2: CMS, same as 0, fix Eroe=Ecms/2
- **3: CMS, use only energy and momentum of signal side**
- 4: CMS, same as 3, update with direction of ROE momentum
- 5: LAB, use energy and momentum of charged particles and photons from whole event
- 6: LAB, same as 5, fix Emiss=$p$miss
- 7: CMS, correct pmiss 3-momentum vector with factor alpha so that dE=0 (used for Mbc calculation).



More resources

- Stuck in your analysis, ask questions at: https://questions.belle2.org/
- Chat with other analysts (experts): https://chat.belle2.org/ or https://b2rc.kek.jp/
- Trouble understanding Physics: The Belle II Physics Book (https://arxiv.org/pdf/1808.10567)

- Again, very detailed basf2 documentation: https://software.belle2.org/
  - Modular analysis: https://software.belle2.org/light-2403-persian/sphinx/analysis/doc/MAWrappers.html
  - List of variables: https://software.belle2.org/light-2403-persian/sphinx/analysis/doc/Variables.html

- Previous years Hands-on session on basf2:
  - 2023 – Boyang Zhang
  - 2022 – Frank Meier

Sample decay file for MC data generation

$B \to D^* e \nu_e$

```
yesPhotos

Decay Upsilon(4S)
1.0  B0sig anti-B0sig B0 anti-B0  VSS_BMIX dm;
Enddecay

Decay anti-B0sig
1.0  D*+sig e- anti-nu_e  BGL 0.02596 -0.06049 0.01311
0.01713 0.00753 -0.09346;
Enddecay
CDecay B0sig

Decay D*+sig
1.0  D0sig pi+  VSS;
Enddecay
CDecay D*-sig

Decay D0sig
1.0  K- pi+  PHSP;
Enddecay

CDecay anti-D0sig

End
```

Sample steering file for MC data generation

```
from basf2 import Path, process, conditions, statistics
from generators import add_evtgen_generator
from simulation import add_simulation
from background import get_background_files
from reconstruction import add_reconstruction
from mdst import add_mdst_output
from glob import glob

main=Path()

decFile='b2dsenu_SM.dec'
output='B2Dslnu_SM_100k.root'

# background (collision) files
bg = glob('/group/belle2/dataprod/BGOverlay/early_phase3/release-06-00-05/overlay/BGx1/set0/*.root')

conditions.prepend_globaltag("mc_production_MC15ri_a")

   main.add_module("EventInfoSetter", expList=[1003], runList=[0], evtNumList=[100000])

# Add the generator
add_evtgen_generator(path=main, finalstate='signal', signaldecfile=decFile)

# Simulate the detector response
add_simulation(path=main, bkgfiles=bg)

# Reconstruct the objects
add_reconstruction(path=main)

# Create the mDST output file
add_mdst_output(path=main, filename=output)

# Process the steering path
process(path=main)
```