Hands-on: basf2

Frank Meier

Belle II Summer Workshop
Iowa
1 – 5 August 2022

Introduction

- basf2: Belle Analysis Software Framework 2
- C++ modules operating on data objects
- python interface to call modules
- `Path` defines sequence of modules to be run
- processing python steering file: `basf2 mysteeringfile.py`
- useful command line arguments (overwrite internal settings in steering file)
  - `--dry-run` checks that syntax is correct and all functions are known but doesn't start event loop
  - `-n 100` limits the event loop to 100 events
  - `-i myinputfile` provides input data file
  - `--help` prints full list of possible arguments
- documentation: https://software.belle2.org

Software releases

- major releases (`release-06`, `release-07`)
  - once a year
  - very thorough validation
  - contains all software changes that are merged to the main branch (after approval of librarian)
- minor releases (`release-06-01`)
  - frequency: one to two for each major release
  - limited amount of new features, usually for specific purpose
- patch releases (`release-06-00-14`)
  - mostly for bug fixes, especially for data-taking and calibration
  - during data-taking synchronized with maintenance days
- light releases (`light-2205-abys`)
  - for introduction of new data analysis features
  - contain only framework, mdst, mva, analysis, skim, geometry, online_book, and b2bii packages
  - no unpacking or digitization ⇒ only mdst and udst can be processed

Basic steering file

- import `basf2` and `modularAnalysis` (optionally with short names)
- create path, *e.g.* `main = b2.Path()`
- read input mdst or udst data using `inputMdst` / `inputMdstList`
- create lists of final-state particles using `fillParticleList`
- form composite particles using `reconstructDecay`
- save variables in output ntuple using `variablesToNtuple`
- `process` the path

Hands-on for three different levels of difficulty

- input mdst file: `~fmeier/BelleIISummerSchool2022/MC15ri_b_ccbar.mdst.root`
- easy
  - reconstruct $D^0 \to K^- \pi^+$
  - save invariant D^0 mass and whether candidate is signal in output ntuple
  - plot invariant D^0 mass distribution of all candidates and split by signal/background
- intermediate
  - reconstruct $D^{*+} \to D^0 \pi^+$ with $D^0 \to K^- \pi^+$
  - apply reasonable, loose selection using PID ((`kaonID` or `binaryPID(211,321)`), distance to IP (`dz`, `dr`), and other kinematic or geometric information (`inCDCAcceptance`, `massDifference(0)`, ...)
  - perform vertex fit of decay chain (TreeFit) with mass constraint on D^0 and IP constraint
  - save multitude of variables for all particles of decay chain in output ntuple including invariant mass of D^0 before the fit

Hands-on for three different levels of difficulty

- advanced
  - input mdst file: `~fmeier/BelleIISummerSchool2022/advanced.mdst.root`
  - find out what kind of decays are present in the file
  - reconstruct signal decay
  - run hadronic FullEventInterpretation
  - combine signal + tag-side to form $\Upsilon(4S)$
  - build rest of event and require no remaining tracks
  - perform best-candidate selection
  - save variables for continuum suppression
- bonus
  - try to run `/home/belle2/fmeier/BelleIISummerSchool2022/faultySteeringScript.py` and fix the errors

Backup

Analysis Software Tools

- particles + particle lists
- decay string grammar
- variables + variable collections + aliases
- ROE
- flavor tagging
- vertex fitting
- best candidate selection
- EventKinematics, EventShape, continuum suppression
- FEI
- documentation: https://software.belle2.org

Data

- dst, cdst, mdst, udst
- `inputMdst(filename, path)`
- `inputMdstList(filelist, path)`
  - `filename / filelist`: path to root input file(s)
  - `environmentType`: optional argument set by default to “default” (mainly for backward compatibility)

Particles

mdst source	particle type
Track	e, μ, π, K, p, d
neutral ECLCluster	γ, K_{L}^{0}, n
neutral KLMCluster	K_{L}^{0}, n
MCParticle	all final state particles
V0	converted γ, K_{S}^{0}, Λ, $\overline{\Lambda}$

- ParticleLoader module creates `Particle` from mdst object
- mdst object $\neq$ `Particle`
  - multiple particles from same mdst object, e.g. track with different mass hypotheses
- mdst source: `isFromECL`, `isFromKLM`, `isFromTrack`, `isFromV0`

ParticleLists

- `fillParticleList('pi+:all', cut="", path)` creates two `ParticleLists`:
  `'pi+:all'` with all positively charged pions and `'pi-:all'` with all negatively charged pions
  - for flavored particles charge-conjugated list always created and filled as well
  - use `"charge > 0"` to select specific flavor (or `"daughter(0, charge) > 0"` for Λ)
    - even `fillParticleList('pi+:negative', 'charge < 0', path)` works
- what's the difference to `fillParticleList('K-:all', cut="", path)`
  - each track fitted with up to six mass hypotheses (at least one fit must have converged)
  - TrackFitResult with mass closest to requested one used
  - hypothesis of used track fit: variable `trackFitHypothesisPDG`
    - cut on this variable or use argument `"enforceFitHypothesis=True"` of `fillParticleList` if you want only a specific mass hypothesis of track fit
  - `pidIsMostLikely` tells whether used mass hypothesis has highest likelihood

⚠ label `all` protected $\rightarrow$ no cut can be applied to these lists

- ECL cluster reconstructed with two different particle hypotheses: photon and neutral hadron
  - crystals considered for cluster might differ
  - ParticleLoader automatically uses correct hypothesis based on particle type

Standard particle lists

- `stdKshorts(prioritiseV0=True, fitter='TreeFit', path)` creates `K_S0:merged` and `stdLambdas(prioritiseV0=True, fitter='TreeFit', path)` creates `Lambda0:merged`
  - vertex fit methods “KFit”, “TreeFit”, and “Rave” available
- `stdXi(fitter='TreeFit', path)` creates `Xi-:std`
- `stdOmega(fitter='TreeFit', path)` creates `Omega-:std`
- `stdMostLikely(path)` creates particle lists for e, μ π, K, p labeled `:mostlikely`
  - internally cut `"thetaInCDCAcceptance and nCDCHits>20"` applied (can be overwritten)
- `stdPi0s(listtype='eff60_May2020', path)` creates π^0 list with 60 % signal efficiency
  - check recommendations of physics performance groups on confluence (*e.g.* here)
- no recommended predefined standard particle lists for charged hadron final state particles

Standard lepton particle lists

- `stdCharged:stdLep` or `stdCharged:stdE` and `stdCharged:stdMu`
  - functions return tuple of alias for letpon ID variable and list of aliases for Data/MC correction weights
- valid working points (values for argument `working_point` / `listtype`)
  - FixedThresh{05, 09, 095}: cut on lepton PID variable
  - UniformEff{60, 70, 80, 90, 95}: uniform efficiency in bins of momentum, polar angle, and charge
- two PID methods: `likelihood` or `bdt`
- two classifications: `binary` (against π hypothesis) or `global` (against all six charged particle hypotheses)
- global tag with payloads for Data/MC correction weights (`lid_weights_gt`), currently `leptonid_Moriond2022_Official_rel5_v2b`
- definition of PID variables differs between release 5 and 6: specify `release`

Combining particles

- `reconstructDecay(decayString, cut, path)` with `Particles` as input
- decay string follows specific decay string grammar
- charge conservation enforced (can be turned off)
- charge-conjugated mode reconstructed as well (switch to turn it off)
- ParticleCombiner ensures that no particle is used twice in same decay chain

⚠ indistinguishable particles per default have different kinematic distributions

e.g. momentum of first π^+ in `D0 -> K- pi+ pi+ pi-` higher than of second π^+

  - shuffle input list randomly to fix this `rankByLowest('pi+:all', 'random', path=path)`

- set argument `dmID` to distinguish different decay modes (access via `extraInfo(dmID)`)

Decay string grammar

- syntax: “mother particle” arrow “daughter particle(s)” `D0 -> K- pi+`
- intermediate decay processes in square brackets `D*+ -> [D0 -> K- pi+] pi+`
- decay string arrows
  - default: `->`
    - accepts intermediate resonances and radiated photons
  - `=direct=>`
    - do not consider intermediate resonances in MC matching
  - `=norad=>`
    - do not consider radiated photons in MC matching
  - `=exact=>`
    - consider neither intermediate resonances or radiated photons in MC matching
  - different arrows allowed in same decay string `D*+ -> [D0 =direct=> K- pi+ pi0] pi+`

Standard variables

- distance to (0, 0, 0) vs distance to IP of reconstructed / generated decay or production vertex
  - `(d0, z0)` vs `(dr, dz)`
  - for MC: `(mcDecayVertexRho, mcDecayVertexZ)` vs `(mcDecayVertexFromIPRho, mcDecayVertexFromIPZ)` and `mcProdVertexX` vs `mcProductionVertexFromIPX`
  - ⚠ `dr`, `mcDecayVertexRho`, and `mcDecayVertexFromIPRho` are magnitudes, all other variables are signed
- polar angle range covered by CDC: `thetaInCDCAcceptance`
- number of CDC hits: `nCDCHits`
- particle identification: `electronID`, `pionID`, etc.
- invariant mass and distance from nominal mass: `M` and `dM`

Bremsstrahlung correction

- recovery of photons emitted by charged particles in magnetic field, especially electrons
- only for tracks most likely to be electrons and with momentum smaller than 5 GeV/c
- extrapolation of track based on VXD hits to ECL
- find nearby neutral clusters and set weights based on angular distance

$$\max\left(\frac{|\phi_{\text{cluster}} - \phi_{\text{hit}}|}{\Delta\phi_{\text{cluster}} + \Delta\phi_{\text{hit}}}, \frac{|\theta_{\text{cluster}} - \theta_{\text{hit}}|}{\Delta\theta_{\text{cluster}} + \Delta\theta_{\text{hit}}}\right)$$

- `correctBrems(outputList, inputList, gammaList, maximumAcceptance=3.0, path)`
  - input and output lists have to be of the same type, typically electrons
  - gamma list has to be defined beforehand
  - per default at most one photon added to a track and each photon only used once
- particles in output list have extraInfo whether a photon has been added (`bremsCorrected`) and extraInfo with energy sum of added photon(s) (`bremsCorrectedPhotonEnergy`)

Marker in decay strings

- `^` : selection of succeeding particle
- `@` : succeeding particle is unspecified, useful for inclusive reconstructions
- `...` : further massive particles in decay mode possible
- `?nu` : decay mode might contain a neutrino
- `?gamma` : decay mode might contain radiative photons
- `?addbrems` : decay mode might contain bremsstrahlung photons
- `(misID)` : succeeding particle is allowed to be other particle type
- `(decay)` : succeeding particle might have decayed in flight, e.g. $\pi \to \mu\nu_\mu$

Variable collections and aliases

Duke UNIVERSITY

- predefined lists of variables loaded via variables.collections
  - cluster, dalitz_3body, deltae_mbc, event_level_tracking, event_shape, extra_energy, flight_info, inv_mass, kinematics, klm_cluster, mc_flight_info, mc_kinematics, mc_tag_vertex, mc_truth, mc_variables, mc_vertex, momentum_uncertainty, pid, reco_stats, recoil_kinematics, roe_multiplicities, tag_vertex, track, track_hits, vertex
- `addAlias('myAliasName', 'real variable name')` part of variables.variables
- `create_aliases(list_of_variables, wrapper, prefix)` part of variables.utils
  - `cmscoll = vu.create_aliases(vc.kinematics, 'useCMSFrame({variable})', 'CMS')`
- `create_aliases_for_selected(list_of_variables, decay_string)` part of variables.utils
  - `vu.create_aliases_for_selected(['M'], 'B0 -> ^J/psi ^K_S0')`
- tutorial on aliases here

Rest of event

- three disjoint group of particles in event: signal + ROE + missing / invisible
- `buildRestOfEvent(target_list_name, path)`
  - default lists to create ROE: all tracks with pion hypothesis, all neutral ECL cluster, all K_L^0 from KLM
  - option to provide own input lists with other than pion hypothesis (`"inputParticlelists=[]"`)
  - argument `"fillWithMostLikely=True"` to use most likely particle hypothesis for each track
- building ROE necessary for flavor tagging and continuum suppression modules
- `fillParticleListFromROE(decayString, cut, path)` creates ROE particle
  - ROE had to be built beforehand
  - mask name can be provided
  - argument `"useMissing=True"` creates `Particle` from missing four-momentum
  - all standard variables can be called for ROE particle

Rest of event masks

- creating masks for selection of charged particles, photons, and neutral hadrons
  - `appendROEMask(list_name, mask_name, trackSelection, eclClusterSelection, path)`
  - `appendROEMasks(list_name, mask_tuples, path)`
- updating existing masks
  - `updateROEMask(list_name, mask_name, trackSelection, eclClusterSelection, path)`
- replacing particles in ROE mask with $V0$ mother

```
fillParticleList('pi+:roe', 'isInRestOfEvent == 1', path = roe_path)
reconstructDecay('K_S0:roe -> pi+:roe pi-:roe', '0.45 < M < 0.55', path = roe_path)
optimizeROEWithV0('K_S0:roe', ['cleanMask'], '', path=roe_path)
mainPath.for_each('RestOfEvent', 'RestOfEvents', path = roe_path)
```

Flavor Tagging

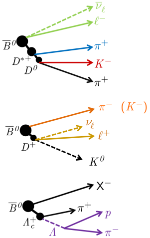
dedicated talk on Friday

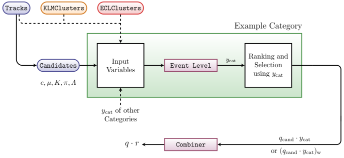
- $\Upsilon(4S) \to B^0\overline{B}^0$ with quantum entanglement of $B^0\overline{B}^0$ pair
- flavor of one of the mesons at its decay determines flavor of other B meson
- centrally trained mva available

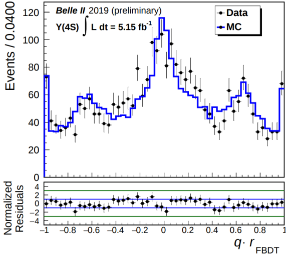
```
buildRestOfEvent(target_list_name, path)
flavorTagger(particleLists, path)
```

- variable collection `ft.flavor_tagging` provides $q \cdot r$ for each category and for combination
- newly introduced argument `maskName=""` to apply selection to ROE

Categories	Targets
Electron	e^-
Intermediate Electron	e^+
Muon	μ^-
Intermediate Muon	μ^+
KinLepton	e^-
Intermediate KinLepton	ℓ^+
Kaon	K^-
KaonPion	K^-, π^+
SlowPion	π^+
FastHadron	π^-, K^-
MaximumP	ℓ^-, π^-
FSC	ℓ^-, π^+
Lambda	Λ
Total= 13	







Fitter

Duke UNIVERSITY

- TreeFit (`vertex.treeFit(list_name, path)`)
  - global fitting tool based on Kalman filter
  - best for complex decay chains, especially when involving long-lived neutrals
- KFit (`vertex.KFit(list_name, conf_level, path)`)
  - fast, simple, kinematic fitter
- RAVE
  - deprecated, but still used in a few places
- OrcaKinFit (`kinfit.fitKinematic4C(list_name, path)` , ...)
  - for over-constrained systems including missing (unmeasured) particles
- TagV (`vertex.TagV(list_name, path)`)
  - vertex fit of tag side using tracks from ROE
  - argument `fitAlgorithm` to select `"Rave"` (default) or `"KFit"`
  - argument `constraintType`: `"IP"` (default), `"tube"`, `"noConstraint"`
  - argument `trackFindingType`: default is to use all tracks with at least one PXD hit, alternatives are to drop PXD criterion (`"standard"`) or use only best track (`"singleTrack"`)

Fit configurations

- update of daughters
  - copies of all daughter particles are created
  - after a successful (converged) fit copies' four-momenta, vertex positions, and covariance matrices updated
  - add `variablesToExtraInfo(particleList, variables, path)` before fit to save pre-fit status
- constraints
  - mass
    - invariant mass constrained (**not fixed**) to PDG value $\Rightarrow$ competes with other constraints of fit
  - IP
    - additional information for initial vertex position (might involve Gaussian smearing or tube)
  - Btube
    - one B selected (in SL decays usually tag-side, in time-dependent analyses usually signal side)
    - other B's direction constrained

TreeFit

- always fits entire decay tree
- internally uses geometric and kinematic constraints
- `massConstraint` accepts pdg code or particle name, applied to all occurrences in all generations of tree
- head of decay chain can be constrained to IP with `ipConstraint=True`

KFit

- various `fit_type` options
  - `"mass"`, `"vertex"` (default), `"massvertex"`, `"fourC"`
- `"ipprofile"` and `"iptube"` as additional constraint options
  - argument `smearing` sets width of IP tube in cm
- works for at most one π^0 in decay mode
- can be used for fit of inclusive particles
- combine p-values of multiple stages of KFits with variable `pValueCombination(p1, p2, …)`

$$p_{\text{combined}} = p_{\text{product}} \cdot \sum_{i=1}^{N} \frac{\left(-\ln p_{\text{product}}\right)^i}{i!} \quad \text{with} \quad p_{\text{product}} = \prod_{j=1}^{N} p_j$$

Best candidate selection

- reconstruction of multiple candidates in same event
  - candidates might share particles, *e.g.* $D^{*\pm} \to D^0\pi^\pm$ with same D^0 but different slow pion
- order candidates based on certain quantity
  - `random`, `abs(dM)`, `chiProb`, ...
  - `rankByHighest(particleList, rankingVariable, path)` or `rankByLowest`
- `allowMultiRank=True` vs. first-come, first-served

⚠ `allowMultiRank=True` in combination with `numBest≠0`

  - `numBest=1` : first multiple candidate kept, all others rejected
  - `numBest=2` : all multiple candidates with best quantity + first of those with second best value kept

MC matching

- in reconstruction weighted relations set between mdst objects (`Track`, `ECLCluster`, `KLMCluster`) and `MCParticle`
- calling `matchMCTruth(B+, path)` in one's steering file sets relations between `Particles` and `MCParticles`
  - recursive matching of all daughter particles
  - bit-wise error flags indicate what went wrong in MC matching (variable `mcError`)

c_Correct = 0	This Particle and all its daughters are perfectly reconstructed.
c_MissFSR = 1	A Final State Radiation (FSR) photon is not reconstructed (based on MCParticle::c_IsFSRPhoton).
c_MissingResonance = 2	The associated MCParticle decay contained additional non-final-state particles (e.g. a rho) that weren't reconstructed. This is probably O.K. in most cases.
c_DecayInFlight = 4	A Particle was reconstructed from the secondary decay product of the actual particle. This means that a wrong hypothesis was used to reconstruct it, which e.g. for tracks might mean a pion hypothesis was used for a secondary electron.
c_MissNeutrino = 8	A neutrino is missing (not reconstructed).
c_MissGamma = 16	A photon (not FSR) is missing (not reconstructed).
c_MissMassiveParticle = 32	A generated massive FSP is missing (not reconstructed).
c_MissKlong = 64	A Klong is missing (not reconstructed).
c_MisID = 128	One of the charged final state particles is mis-identified (wrong signed PDG code).
c_AddedWrongParticle = 256	A non-FSP Particle has wrong PDG code, meaning one of the daughters (or their daughters) belongs to another Particle.
c_InternalError = 512	There was an error in MC matching. Not a valid match. Might indicate fake/background track or cluster.
c_MissPHOTOS = 1024	A photon created by PHOTOS was not reconstructed (based on MCParticle: :c_IsPHOTOSPhoton).
c_AddedRecoBremsPhoton = 2048	A photon added with the bremsstrahlung recovery tools (correctBrems or correctBremsBelle) has no MC particle assigned, or it doesn't belong to the decay chain of the corrected lepton mother

MC matching examples

- sample with
  a) $D^0 \to K^-\pi^+\pi^0$
  b) $D^0 \to K^-\rho^+$ with $\rho^+ \to \pi^+\pi^0$
  c) $D^0 \to K^-\pi^+\pi^0\gamma$
  d) $D^0 \to K^{*-}\pi^+$ with $\pi^+ \to \mu^+\nu_\mu$
  e) $D^0 \to K^{*-}\mu^+\nu_\mu$

decay string	`isSignal == 0`
`D0 -> K- pi+ pi0`	a), b), c)
`D0 =direct=> K- pi+ pi0`	a) and c)
`D0 =exact=> K- pi+ pi0`	only a)
`D0 -> K- (decay)pi+ pi0`	a), b), c), and d)
`D0 -> K- (decay)pi+ pi0 ?nu`	a), b), c), d), and e)
`D0 -> (misID)pi- pi+ pi0`	a), b), c)

MC matching variables and MC particle lists

- `isSignal`: generated decay is correctly reconstructed according to decay string grammar
- `?addbrems` in decay string: `isSignal` behaves like `isSignalAcceptBremsPhotons`
- `?nu` in decay string: `isSignal` behaves like `isSignalAcceptMissingNeutrino`
- `...` in decay string: `isSignal` behaves like `isSignalAcceptMissingMassive`
- `mc_gen_topo()`: variable collection for TopoAna tool
- create MC particle lists using `fillParticleListFromMC(decayString, cut, path)`
  - add argument `"addDaughters=True"` to recursively create particles for all daughters and set relation to mother MC particle
  - variable `isMCDescendantOfList` allows to figure out relatives
- dedicated `reconstructMCDecay(decayString, cut, path)` with MC particles as input

EventKinematics vs EventShape

- calculate global kinematics of event: `buildEventKinematics`
  - visible energy, total photon energy, missing momentum (in lab and CMS frame)
  - track's mass hypothesis matters
    - can use argument `fillWithMostLikely` to use most likely hypothesis for each track
- calculate event-level shape quantities: `buildEventShape(path)`
  - cleo cones, collision axis, fox wolfram moments, harmonic moments, jets, sphericity, thrust
  - apart from jet calculation mass hypothesis of tracks irrelevant
- standard cuts on tracks and photons
  - p_{T} `> 0.1`, `-0.866 <` $\cos\theta$ `< 0.9535`, `dr < 0.5`, `|dz| < 3`
  - `E > 0.05`, `-0.866 <` $\cos\theta$ `< 0.9535` (CDC acceptance)
- one can provide own `inputListNames` but then need to apply selection cuts yourself

⚠ duplicates in input lists distort true distributions

Weights

- ParticleWeightingLookUpCreator module (tutorial B2A904-LookUpTableCreation)
  - define experiment and run range, table name, (multi-dimensional) binning of variables
  - set weight and errors of any kind for each bin as dictionary
- ParticleWeighting module (tutorial B2A905-ApplyWeightsToTracks)
  - provide `particleList` and `tableName`
  - requires `ParticleWeightingLookupTable` to be present in conditions database
  - adds `extraInfo`(s) to particles
- `variablesToEventExtraInfo(particleList, variables, path)`
  - adds (candidate- or event-based) quantities to `eventExtraInfo`
  - works like `variablesToExtraInfo`
  - original intended use-cases
    - best candidate selection among different particle lists, *e.g.* B^+ vs B^0
    - relate MC information to reconstructed candidates

Best candidate selection example

- scenario: multiple D^0 candidates, multiple π^+ candidates $\Rightarrow$ many $D^{*+} \to D^0\pi^+$ candidates
- plan: first select D^{*+} with higher momentum π^+, then if necessary D^{*+} with better D^0 vertex fit quality
  - `variables.addAlias('PiMomentum', 'daughter(1, p)')`
  - `rankByHighest('D*+', 'PiMomentum', allowMultiRank=True, numBest=0, path=main)`
  - `applyCuts('D*+', 'extraInfo(PiMomentum_rank) == 1', path=main)`
  - `variables.addAlias('D_chiProb', 'daughter(0, chiProb)')`
  - `rankByHighest('D*+', 'D_chiProb', allowMultiRank=False, numBest=1, path=main)`

B2BII

- special particle lists for neutrals
  - `gamma:mdst`, `pi0:mdst`, `K_S0:mdst`, `Lambda0:mdst`, `gamma:v0mdst` (converted photons), `K_L0:mdst`
- dedicated PID variables
  - `atcPIDBelle()`, `eIDBelle`, `muIDBelle`, `muIDBelleQuality`
- "standard cuts" for K_S^0 and Λ: `goodBelleKshort` and `goodBelleLambda`