US Belle II Summer School: BASF2 Analysis

Jake Bennett
Carnegie Mellon University
August 16, 2016

Part 3: Basics of basf2 analysis

- Setting up for an analysis
- Standard analysis tools
- A simple example

Release setup

- Let's get started!
- We will use the central basf2 release again today

- Remember to configure the environment in each new shell!
  - If you are on KEKCC,

```
[user ~]$ source /sw/belle2/tools/setup_belle2
[user ~]$ setuprel release-00-07-01
```

- If you set up an analysis repository

```
[user ~]$ cd Tutorials-08-2016
[user ~]$ setupana
```

Remember, your analysis repository is linked to a central release!

BASF2 Analysis software

- Similar to the framework, the analysis software in basf2 is modular
  - Belle II analyses should be intuitive and standardized
  - Take a look in the analysis package in the scripts directory

```
[user ~]$ ls $BELLE2_RELEASE_DIR/analysis/scripts/
analysisPath.py        stdCharm.py              stdPhotons.py
fei                    stdDiLeptons.py          stdPi0s.py
FlavorTagger.py        stdFSParticles.py        stdV0s.py
modularAnalysis.py     stdLightMesons.py        variables.py
stdCharged.py          stdLooseFSParticles.py   vertex.py
```

- Many useful methods are defined in scripts like these
- For example: add_simulation, add_reconstruction, inputMdst, add_mdst_output, reconstructDecay, etc.

Example analysis method (in modularAnalysis.py)

```
def reconstructDecay(
    decayString,
    cut,
    dmID=0,
    writeOut=False,
    path=analysis_main,
):
    """
    Creates new Particles by making combinations of existing Particles - it reconstructs unstable
    particles via their specified decay mode, e.g. in form of a DecayString: D0 -> K- pi+;
    + -> anti-D0 pi+, .... All possible combinations are created (overlaps are forbidden) and
    combinations that pass the specified selection criteria are saved to a newly created (mother)
    ParticleList. By default the charge conjugated decay is reconstructed as well (meaning that
    the charge conjugated mother list is created as well).

    @param decayString DecayString specifying what kind of the decay should be reconstructed
                       (from the DecayString the mother and daughter ParticleLists are
determined)
    @param cut         created (mother) Particles are added to the mother ParticleList if they
                       pass give cuts (in VariableManager style) and rejected otherwise
    @oaram dmID        user specified decay mode identifier
    @param writeOut    wether RootOutput module should save the created ParticleList
    @param path        modules are added to this path
    """
```

call this in your analysis steering file

the result is a list of Particles (note the capitalization)

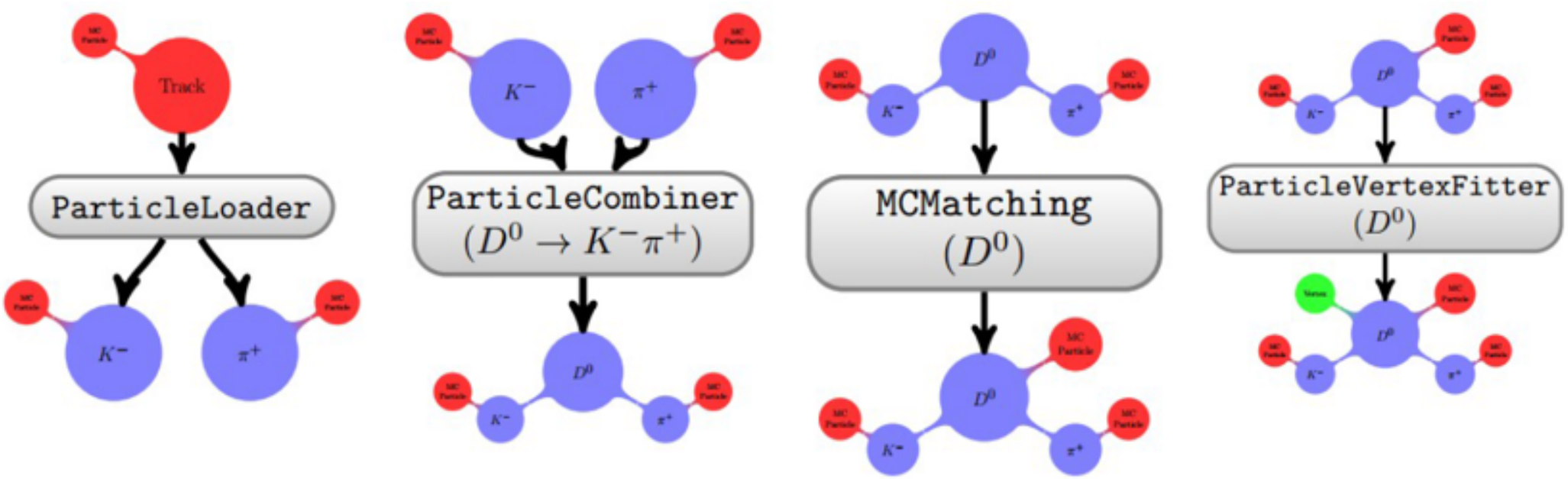
Example analysis method (in modularAnalysis.py)

- In the analysis chain, one or more modules are executed (eg. ParticleCombiner), each with a specific purpose

```
pmake = register_module('ParticleCombiner')
pmake.set_name('ParticleCombiner_' + decayString)
pmake.param('decayString', decayString)
pmake.param('cut', cut)
pmake.param('decayMode', dmID)
pmake.param('writeOut', writeOut)
path.add_module(pmake)
```

the parameters from your steering file get passed to the module

- At the end of the analysis chain, a list of Particles exist in the DataStore



mDST data objects: Particles

- Analysis sequencing and objects are built around Particles and their containers (ParticleLists)
  - Minimal representation of the "physical" particle
  - 4-momentum, position (vertex), error matrix, PDG code, Particle type, daughters, …
  - e, μ, π, K, p, d (Tracks), γ (ECLClusters), K_L (KLMClusters)

- Low-level analysis tools:
  - ParticleLoader - creates Particles from mDST objects
  - ParticleSelector - applies cuts on existing Particles
  - ParticleCombiner - performs combinations
  - …

- More details on the analysis confluence page

Analysis descriptors

- Analysis modules use the DecayString (borrowed from LHCb)
- Describes the structure and contents of a decay chain
- The framework translates particle names to PDG codes
- You can probably guess the meaning of this DecayString

```
'B+ -> [anti-D0 -> K- pi+] pi+'
```

- Inclusive decays are specified by an ellipsis (...), i.e. for missing/unreconstructed particles
- Select particles with a carrot (^) for writing out information

```
'B+ -> [anti-D0 -> ^K- ^pi+] pi+'
```

yes yes no

Writing out information

- The properties of the Particle objects from the DataStore may be written to an Ntuple using the NtupleMaker

```
…

from modularAnalysis import ntupleFile
from modularAnalysis import ntupleTree

ntupleFile('B2D0pi-output.root')

…

NtupleTools = ['NtupleToolName', 'DecayString']
NtupleTools += ['AnotherNtupleToolName', 'AnotherDecayString']

…

ntupleTree('TreeName', 'ParticleList', NtupleTools)

…
```

create the file

only information for ***selected*** particles is saved

save the info to an Ntuple

Writing out information

- Quick Exercise: Take a look at the available NtupleTools by running the variables.py steering file
  'basf2 $BELLE2_RELEASE_DIR/analysis/scripts/variables.py'

- You can create new NtupleTools (or request one)
- A generic tool to write out any available analysis variable is called CustomFloats

Example 4: A simple exclusive MC analysis

From your analysis repository (maybe in a directory called 'examples')

a) Write a decay.dec file
   - Specify $c\bar{c}$ continuum events, where any D^{*+} produced must decay to π^+D^0 (and c.c.) and the D^0 must decay to $K^-\pi^+$

b) Write a steering script to generate 10 continuum D* events

c) Check the contents of the output ROOT file

d) Simulate and reconstruct the events

e) Write a script to analyze the reconstructed events

4. (a) Write the decay.dec file (name it exercise4.dec)
   - Specify $c\bar{c}$ continuum events, where any D^{*+} produced must decay to $\pi^+ D^0$ (and c.c.) and the D^0 must decay to $K^-\pi^+$

4. (a) Write the decay.dec file (name it exercise4.dec)

- Specify $c\bar{c}$ continuum events, where any D^{*+} produced must decay to $\pi^+ D^0$ (and c.c.) and the D^0 must decay to $K^-\pi^+$

```
#be sure to include JetSetParameters (see next slide)

Alias MyD0 D0
Alias MyAntiD0 anti-D0
ChargeConj MyD0 MyAntiD0

Decay vpho
# Pythia6 definition "32" for "vpho -> q qbar" should be used.
#   (this is automatically converted to Pythia8 definition "91")
1.000 c anti-c      PYTHIA 32;
Enddecay

Decay D*-
1.000 MyAntiD0 pi-   VSS;
Enddecay

Decay D*+
1.000 MyD0 pi+       VSS;
Enddecay

Decay MyD0
1.000 K- pi+        PHSP;
Enddecay
CDecay MyAntiD0

End
```

Create an alias for decay of D^0 from D^*

Specify antiparticles for D^* daughters

Ask for $c\bar{c}$ decay

Force every D^{*+} to decay to $D^0\pi^+$ and c.c.

Force every D^0 from a D^* to decay to $K^-\pi^+$ and c.c

JetSet parameters
(from analysis/examples/tutorials/B2A102-ccbarEventGeneration.dec)

```
#JetSetPar PARJ(21)=0.28 ---> Pythia8 does not support this parameter, but a similar Pythia8 parameter may be available

#JetSetPar PARJ(25)=0.27 ---> corresponding to Pythia8 StringFlav:etaSupPythiaBothParam
StringFlav:etaSup=0.27

#JetSetPar PARJ(26)=0.12 ---> corresponding to Pythia8 StringFlav:etaPrimeSupPythiaBothParam
StringFlav:etaPrimeSup=0.12

#JetSetPar PARJ(33)=0.3 ---> corresponding to Pythia8 StringFragmentation:stopMass
PythiaBothParam StringFragmentation:stopMass=0.3

#JetSetPar PARJ(41)=0.32 ---> corresponding to Pythia8 StringZ:aLund
PythiaBothParam StringZ:aLund=0.32

#JetSetPar PARJ(42)=0.62 ---> corresponding to Pythia8 StringZ:bLund
PythiaBothParam StringZ:bLund=0.62

#JetSetPar PARJ(81)=0.38 ---> Pythia8 does not support this parameter, but a similar Pythia8 parameter may be available
#JetSetPar PARJ(82)=0.76 ---> Pythia8 does not support this parameter, but a similar Pythia8 parameter may be available

#JetSetPar PARP(2)=4.0 ---> Pythia8 does not support this parameter, but a similar Pythia8 parameter may be available
#JetSetPar MSTP(141)=1 ---> Pythia8 does not support this parameter, but a similar Pythia8 parameter may be available
#JetSetPar MSTP(171)=1 ---> Pythia8 does not support this parameter, but a similar Pythia8 parameter may be available
#JetSetPar MSTP(172)=1 ---> Pythia8 does not support this parameter, but a similar Pythia8 parameter may be available

#JetSetPar MSTJ(11)=4 is not supported. Therefore, it is changed from 4 to 5.
#JetSetPar MSTJ(11)=5 ---> corrsponding to Pythia8 StringZ:usePeterson{C,B,H}
PythiaBothParam StringZ:usePetersonC=off
PythiaBothParam StringZ:usePetersonB=off
PythiaBothParam StringZ:usePetersonH=off

#JetSetPar PARJ(46)=1.0 ---> corresponding to Pythia8 StringZ:rFactC
PythiaBothParam StringZ:rFactC=1.0
```

Don’t change these unless you know what you are doing!

4. (b) Write a steering script to generate 10 continuum D* events
(and save the output in mDST format)

4. (b) Write a steering script to generate 10 continuum D* events

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

from basf2 import *
from modularAnalysis import *
from reconstruction import add_mdst_output

# Generate continuum events according to decay.dec
generateContinuum(10, 'D*+', 'exercise4.dec')

# Write out events to mDST file
add_mdst_output(analysis_main, True, 'exercise4b.root')

# Process all modules added to the analysis_main path
process(analysis_main)

# Print out the summary
print(statistics)
```

Make sure to include any necessary methods

Generate continuum events where each event must contain a D*

Write generated events to mDST format

Useful module to print statistics

4. (c) Check the contents of the generated file

- Check the DataStore content

```
[user ~]$ basf2 $BELLE2_RELEASE_DIR/analysis/examples/
printDataStore.py -i exercise4b.root
```

```
[INFO] ==================================================================
[INFO] DataStore collections in event 0
[INFO] ==================================================================
[INFO] Type                 Name                        #Entries          <Event>
[INFO] EventMetaData        EventMetaData
[INFO] MCParticle[]         MCParticles                 22
[INFO]
[INFO] ------------------------------------------------------------------
[INFO] Type                 Name                        #Entries      <Persistent>
[INFO] FileMetaData         FileMetaData
[INFO] ProcessStatistics    ProcessStatistics
[INFO]
[INFO] ==================================================================
```

4. (c) Check the contents of the generated file

- Check the topology of one event

```
[user ~]$ basf2 $BELLE2_RELEASE_DIR/analysis/examples/
printMCParticleArray.py -n 1 -i exercise4b.root
```

```
[INFO]   1   10022 (vpho)          E: 1.101e+01 m: 1.058e+01 p:( 4.568e-01,-0.000e+00, 3.001e+00)
[INFO]   2       421 (D0)          E: 2.718e+00 m: 1.865e+00 p:( 1.509e+00,-1.670e-01,-1.266e+00)
[INFO]   8         310 (K_S0)      E: 6.130e-01 m: 4.976e-01 p:( 9.290e-02,-1.282e-01,-3.210e-01)
[INFO]  17          211 (pi+)      E: 3.177e-01 m: 1.396e-01 p:(-1.350e-01,-1.547e-01,-1.982e-01)
[INFO]  18         -211 (pi-)      E: 2.953e-01 m: 1.396e-01 p:( 2.279e-01, 2.653e-02,-1.228e-01)
[INFO]   9        211 (pi+)        E: 1.223e+00 m: 1.396e-01 p:( 1.098e+00,-3.656e-01,-3.678e-01)
[INFO]  10       -211 (pi-)        E: 5.952e-01 m: 1.396e-01 p:( 3.196e-01, 3.556e-01,-3.258e-01)
[INFO]  11        111 (pi0)        E: 2.869e-01 m: 1.350e-01 p:(-1.798e-03,-2.888e-02,-2.515e-01)
[INFO]  19           22 (gamma)    E: 1.762e-01 m: 9.203e-10 p:(-2.666e-02,-7.826e-02,-1.556e-01)
[INFO]  20           22 (gamma)    E: 1.107e-01 m: 9.203e-10 p:( 2.486e-02, 4.938e-02,-9.590e-02)
[INFO]   3      211 (pi+)          E: 1.063e+00 m: 1.396e-01 p:( 4.093e-01, 3.391e-02,-9.705e-01)
[INFO]   4      113 (rho0)         E: 9.949e-01 m: 8.090e-01 p:( 2.821e-01, 8.476e-02, 4.987e-01)
[INFO]  12        211 (pi+)        E: 2.858e-01 m: 1.396e-01 p:(-1.247e-01,-2.149e-01, 2.124e-02)
[INFO]  13       -211 (pi-)        E: 7.087e-01 m: 1.396e-01 p:( 4.065e-01, 2.998e-01, 4.771e-01)
[INFO]  14         22 (gamma)      E: 4.318e-04 m: 0.000e+00 p:( 2.422e-04,-1.274e-04, 3.340e-04)
[INFO]   5     -211 (pi-)          E: 2.709e-01 m: 1.396e-01 p:( 1.012e-01, 5.827e-02,-2.007e-01)
[INFO]   6      211 (pi+)          E: 1.875e+00 m: 1.396e-01 p:(-2.317e-01,-4.550e-01, 1.799e+00)
[INFO]   7     -413 (D*-)          E: 4.087e+00 m: 2.010e+00 p:(-1.613e+00, 4.450e-01, 3.140e+00)
[INFO]  15       -421 (anti-D0)    E: 3.803e+00 m: 1.865e+00 p:(-1.517e+00, 4.483e-01, 2.912e+00)
[INFO]  21          321 (K+)       E: 2.797e+00 m: 4.937e-01 p:(-6.931e-01, 8.514e-01, 2.525e+00)
[INFO]  22         -211 (pi-)      E: 1.005e+00 m: 1.396e-01 p:(-8.241e-01,-4.031e-01, 3.871e-01)
[INFO]  16       -211 (pi-)        E: 2.843e-01 m: 1.396e-01 p:(-9.616e-02,-3.278e-03, 2.282e-01)
```

fragmentation particles

4. (d) Simulate and reconstruct the events

(read in the mDST output from 4b)

4. (d) Simulate and reconstruct the events

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

# analysis_main is the default path created in the modularAnalysis.py
from basf2 import *
from modularAnalysis import *
from simulation import add_simulation
from reconstruction import add_reconstruction
from reconstruction import add_mdst_output

# Load the input ROOT file
inputMdst('exercise4b.root')

# Simulation and reconstruction
add_simulation(analysis_main)
add_reconstruction(analysis_main)

# Write out events to mDST file
add_mdst_output(analysis_main, True, 'exercise4d.root')

# Process all modules added to the analysis_main path
process(analysis_main)

# Print out the summary
print(statistics)
```

Make sure to include any necessary methods

Read the mDST file

Simulate and reconstruct the events

Write the output to mDST

4. (e) Write a script to analyze the reconstructed events

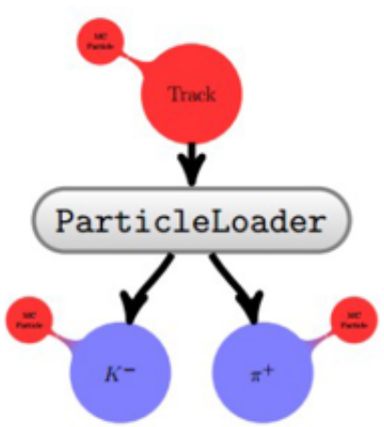
4. (e) Write a script to analyze the reconstructed events

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

from basf2 import *
from modularAnalysis import *

# Load input ROOT file
inputMdst('exercise4d.root')

# set the proper magnetic field
analysis_main.add_module('Geometry', ignoreIfPresent=False, components=['MagneticField'])

# Create and fill final state ParticleLists
fillParticleList('pi+:std', 'chiProb > 0.001 and abs(d0) < 2 and abs(z0) < 4')
fillParticleList('K+:std', 'chiProb > 0.001 and abs(d0) < 2 and abs(z0) < 4')

---
```

Read the mDST file

Set the magnetic field (see note on slide 23)

Fill the particle lists



Naming convention: *listName = particleName:label*

look in evt.pdl for names

descriptive string

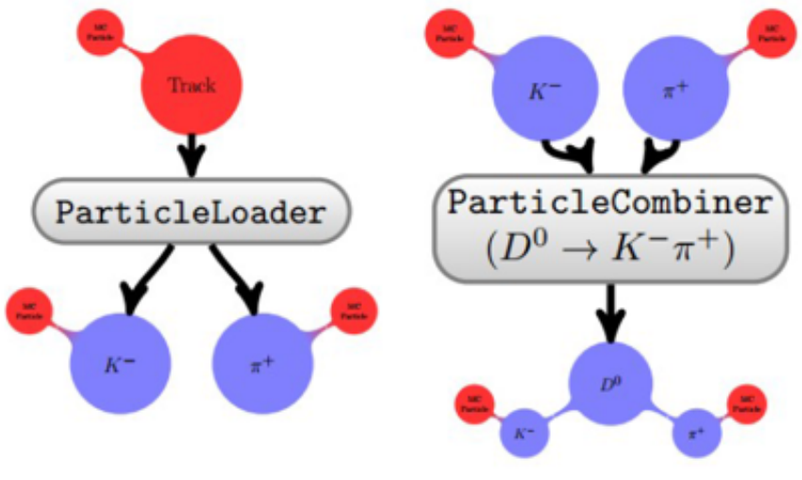
eg. *pi+:loose* or *D0:kpi*

4. (e) Write a script to analyze the reconstructed events

```
---

# Create and fill final state ParticleLists
fillParticleList('pi+:std', 'chiProb > 0.001 and abs(d0) < 2 and abs(z0) < 4')
fillParticleList('K+:std', 'chiProb > 0.001 and abs(d0) < 2 and abs(z0) < 4')

# Reconstruct D0:kpi
reconstructDecay('D0:kpi -> K-:std pi+:std', '1.835 < M < 1.895')

# Reconstruct the D*+ from the D0:kpi and pi+:std
reconstructDecay('D*+:sig -> D0:kpi pi+:std', '1.7 < M < 2.3')

---
```

Reconstruct the D^0

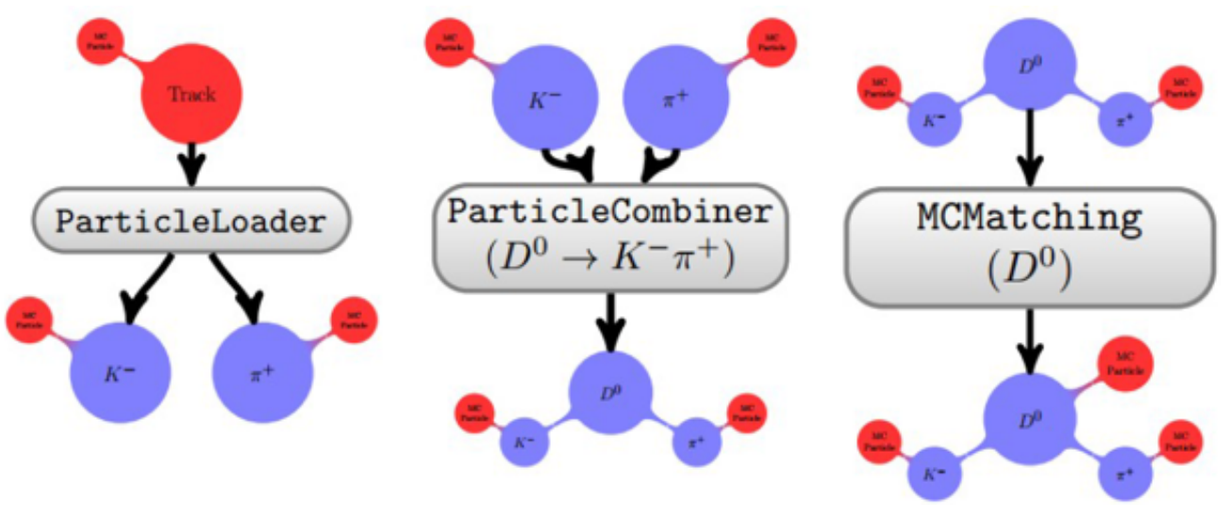
Reconstruct the D*



4. (e) Write a script to analyze the reconstructed events

```
---

# perform MC matching (MC truth asociation)
matchMCTruth('K-')
matchMCTruth('pi+')
matchMCTruth('D*+')

---
```

Use analysis module to match reconstructed particles with MC particles



4. (e) Write a script to analyze the reconstructed events

```
---

# Write out useful information to a ROOT file
# information to be saved to file
toolsdst = ['EventMetaData', '^D*+:sig']
toolsdst += ['RecoStats', '^D*+:sig']
toolsdst += ['Kinematics', '^D*+:sig -> [^D0:kpi -> ^K-:std ^pi+:std] ^pi+:std']
toolsdst += ['InvMass', '^D*+:sig -> [^D0:kpi -> K-:std pi+:std] pi+:std']
toolsdst += ['MCTruth', 'D*+:sig -> [D0:kpi -> ^K-:std ^pi+:std] ^pi+:std']
toolsdst += ['PID', 'D*+:sig -> [D0:kpi -> ^K-:std ^pi+:std] ^pi+:std']

# write out the flat ntuple
ntupleFile('exercise4e.root')
ntupleTree('dsttree', 'D*+:sig', toolsdst)

# Process the events
process(analysis_main)

# Print call statistics
print(statistics)
```

Define the variables to add to an Ntuple

Define the output ROOT file and TTree

Quick Exercise: Take a look at exercise4e.root with the ROOT TBrowser

Don't forget to check in your changes!

- Publish your analysis files to your repository to track modifications and back up your work

```
[user ~]$ git add --all
[user ~]$ git commit -m "commit message"
[user ~]$ git push
```

A note on standard reconstruction lists

- The software is still in flux and some tools become obsolete quite quickly
  - Caution: be careful with the photon or pi0 lists (goodPhoton, stdPi0, etc.)
    - These are broken for various reasons
    - Be sure to check the analysis information on confluence
      https://confluence.desy.de/display/BI/Physics+Pi0Reco
      https://confluence.desy.de/display/BI/Physics+StandardParticles
    - Hopefully this will be fixed in future releases

- For reference, this is what works for me in release-00-07-01

```
---

fillParticleList('gamma:mysel', '[E > 0.072 and clusterE9E25>0.800 and minC2HDist>39 and
clusterReg==1] or [E>0.071 and clusterE9E25>0.805 and minC2HDist>29 and clusterReg==2] or
[E>0.066 and clusterE9E25>0.710 and minC2HDist>23 and clusterReg==3]')
reconstructDecay('pi0:all -> gamma:mysel gamma:mysel', '0.11 < M < 0.15', 1, Tru
e)
massKFit('pi0:all', 0.0, '', analysis_main)

---
```

A note on magnetic field settings (from Anze)

- The default magnetic field map in basf2 has changed form constant field of 1.5 T to 3D map. The former was used to produce MC5 for example, but the latter is loaded when analysing it. Since the fields are quite different this leads to large biases in momenta of charged tracks.
- When performing the analysis over mdst MC(DATA) samples it is important that the environment (for example magnetic field map) is set up correctly. Unfortunately this information is not embeded to mdst files so it is not possible to retrieve it form the file itself (especially in case of microDST files). Instead users are required to specify in their python steering files correct environment for the sample they are analysing. At the moment these are:
  - MC5: this environment should be used when doing the analysis over Belle II MC samples produced prior to build-2016-05-01
  - default: this environment should be used when doing the analysis over Belle II MC samples produced with build-2016-05-01 or newer. This environment contains the current default magnetic field map.
  - Belle: this environment should be used when doing the analysis over converted Belle MC or data.
  - None: in the case you're performing the analysis on generator level the geometry with magnetic field needs not to be loaded.

- The environment is specified via new (obligatory) parameter in 'inputMdst(…)' and 'inputMdstList(…)' modularAnalysis convenience functions. **If you're using rev29858 or going to use coming July build, then you're scripts will not work until you specify the proper environment via the inputMdst(List) argument.**

A note on magnetic field settings (from Anze)

- Of course, this solution only works if you are using a more recent build
- For release-00-07-01 (and prior builds), you must set the parameters yourself if you are going to run over MC5 samples:

```
---

# load input ROOT file
inputMdst(inputfile)

# set the magnetic field to that used in MC5
analysis_main.add_module('Geometry', ignoreIfPresent=False, components=['MagneticFieldConstant'])

---
```

- When using data generated with a build
  - prior to build-2016-05-01 (including MC5), use “MagneticFieldConstant”
  - at or after build-2016-05-01 (including MC6), use “MagneticField”
  - on converted Belle data, use “MagneticFieldConstantBelle”

Additional tools

- Generator level skims
  - If you want to generate continuum MC, but you only care about a particular channel, you can skim at the generator level to same CPU time
  - For example, if we want to specify the fragmentation particles in Ex. 4

```
---

# Select decays based on the number of fragmentation pions
cutAndCopyList('vpho:mode1', 'vpho:all', 'countDaughters(abs(mcPDG) == 413) == 1 and
countDaughters(abs(mcPDG) == 421) == 1 and countDaughters(abs(mcPDG) == 211) == 1 and
countDaughters(1==1) == 3')

# Create empty path
empty_path = create_path()

# Register the skim module, which passes or fails events based on the indicated criteria
skim = register_module('SkimFilter')
skim.param('particleLists', ['vpho:mode1'])
analysis_main.add_module(skim)

# Jump to empty path if event is of the wrong channel
skim.if_false(empty_path)

---
```

Specify daughters of virtual photon

Create an empty path in which to dump events of the wrong type

Use the SkimFilter module to pass or fail events based on the criteria

Dump failed events into the empty path

Additional tools

- Other useful tools (not an exhaustive list!)
  - Reconstruct multiple final states (B2A303)

```
reconstructDecay('D0:ch1 -> K- pi+', '1.8 < M < 1.9', 1)
reconstructDecay('D0:ch2 -> K- pi+ pi0', '1.8 < M < 1.9', 2)
reconstructDecay('D0:ch3 -> K- pi+ pi+ pi-', '1.8 < M < 1.9', 3)
reconstructDecay('D0:ch4 -> K- K+', '1.8 < M < 1.9', 4)
reconstructDecay('D0:ch5 -> pi+ pi-', '1.8 < M < 1.9', 5)

# merge the D0 lists together into one single list
copyLists('D0:all', ['D0:ch1', 'D0:ch2', 'D0:ch3', 'D0:ch4', 'D0:ch5'])
```

  - Vertex and kinematic fitting (B2A40*)

```
vertexKFit('D0:kpi', 0.0)

massKFit('D0:kpi', 0.0)
```

apply cuts here, (C.L. > 0) = no cut

Additional tools

- Other useful tools (not an exhaustive list!)
  - Rank candidate events (B2A602)

```
rankByLowest('D0', 'chiProb')

___

rankByLowest('D0', 'chiProb', 1)
```

only keep (1) event with best χ^2

  - TMVA (B2A50*)
  - Continuum suppression (B2A70*)
  - Flavor tagging (B2A801)
  - Printing DataStore information, etc (B2A90*)

Batch jobs (only at KEKCC)

- From the KEK work server, you can submit a batch job
  - Scheduling is handled by LSF (Platform Load Sharing Facility)
  - Dispatches and executes the batch job on a calculation server
  - When submitting a job, it is necessary to choose a queue that is appropriate
- To submit a job into LSF, run the following command

```
[user ~]$ bsub -q queueName jobName
```

- The output file is “Job-ID.out” under the directory .lsf in your home directory
  - You can change this with the -o option (if only a directory is given, the output is still given by Job-ID.out, but the files are saved here)

Batch jobs (only at KEKCC)

Quick Exercise: Submit exercise1.py to the short queue.

```
[user ~]$ bsub -q s 'basf2 exercise1.py'
```

- You can check on your jobs using "*bjobs*" and kill jobs using "*bkill jobID*"

- More details available in the data analysis system user's guide

Batch jobs (only at KEKCC)

- You can also create a shell file and specify arguments there

```
#!/bin/bash
#
#BSUB -J exercise1
#BSUB -e exercise1.err
#BSUB -o exercise1.out
#BSUB -q s

basf2 exercise1.py
```

job name

error file

output file

queue name

- Then submit the shell script

```
[user ~]$ bsub < exercise1.sh
```

Solutions and additional resources

- For my solutions to the exercises, take a look at my git repository: https://stash.desy.de/projects/B2A/repos/jbennett_tutorials-08-2016/browse

- More examples exist in other basf2 tutorials
  - Introduction to basf2 by Thomas Kuhr and Christian Pulvermacher
  - Analysis Tutorial by Luis Pesantez; Part 2
  - Analysis Software Tutorial June 2016 by Anze Zupanc

You can find more information online

- Belle II page (navigate to software, physics->analysis, etc)
  - https://www.belle2.org
- basf2 Manual
  - https://confluence.desy.de/display/BI/Software+Basf2manual
- basf2 documentation
  - https://belle2.cc.kek.jp/internal/software/
- Mainling lists
  - https://lists.belle2.org/sympa
- Git/stash resources
  - Introduction: https://confluence.desy.de/pages/viewpage.action?pageId=35819226
  - Personal repositories: https://stash.desy.de/users/username