FCC and COFFEA

A fully pythonic workflow for FCCAnalyses

Prayag Yadav ↗, David Lange, Bhawna Gomber
HSF-India

Outline

COFFEA

- **Columnar Object Framework For Effective Analysis (COFFEA)**[1] is a python package that unifies all the necessary HEP scientific python packages for a full analysis.

[1] COFFEA Docs

COFFEA and the Python ecosystem

COFFEA combines various HEP-relevant frameworks to create a unified analysis tool:

- **Uproot** to read and write root files.
- **Awkward Array** to manipulate HEP data in a numpythonic way.
- **Dask** for scaling-out
- **Hist** for histogramming
- **MPLHEP** for plotting
- **Numba** for just-in-time compilation
- and other useful packages ...

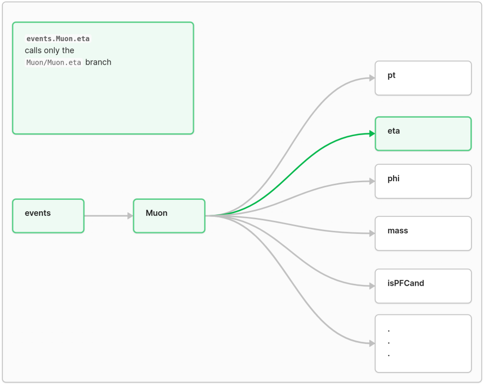
Lazy access model for columnar data processing

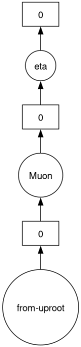
An example in CMS-NanoAOD (COFFEA is widely used in CMS Analysis):

```
test_file = "../coffea-fcc-analyses/data/CMS_Run2018A_MET/0F8C0C8C-63E4-1D4E-A8DF-506BDB55BD43.root"
from coffea.nanoevents import NanoEventsFactory, NanoAODSchema
events = NanoEventsFactory.from_root(
    test_file+":Events",
    schemaclass=NanoAODSchema,
    delayed = True
).events()
events.Muon.fields
```

```
output:
 [
  pt,
  eta,
  phi,
  mass,
  charge,
  isPFCand,
  etc.
  .
  .
  .
 ]
```

Lazily access only the branches required for your computation

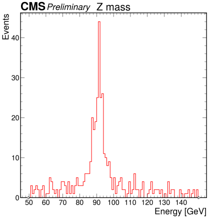


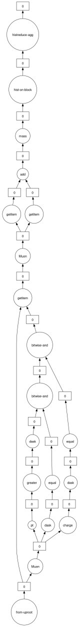


A simple example in CMS NanoAOD

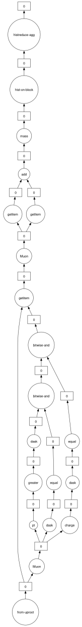
```python
class MyZPeak(processor.ProcessorABC):
    """Get a Z peak mass plot"""

    def __init__(self):
        pass

    def process(self, events):
        muon_pt_gt_25 = ak.all(events.Muon.pt > 25, axis=1)
        exactly_two_muons = ak.num(events.Muon, axis = 1) == 2
        opp_charged_muons = ak.sum(events.Muon.charge, axis = 1) == 0
        dimuon_events = events[muon_pt_gt_25 & exactly_two_muons & opp_charged_muons]
        z = dimuon_events.Muon[:,0] + dimuon_events.Muon[:,1]
        h = hda.Hist.new.Reg(100,50,150).Double()
        h.fill(z.mass)
        return {"Z_mass": h}

    def postprocess(self):
        pass
```

```python
p = MyZPeak()
out = p.process(events)
(computed,) = dask.compute(out)
h = computed["Z_mass"]
```



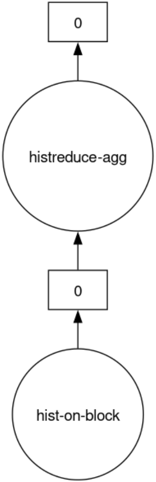


Dask optimizes computations



dask.optimize(h)

Groups together similar operations, determines the necessary parts of a dataset required for the final computation (more info)



A Schema adds format specific functionalities

- COFFEA supports schemas for reading data efficiently:
  - Build collections of branches
  - Add helper functions
  - Describe relations between branches
- The BaseSchema works for any columnar dataformat.
  - Specific schemas developed on top make it easier for users to read data and use it in their analysis

Available schema
BaseSchema
DelphesSchema
FCCSchema
TreeMakerSchema
PHYSLITESchema
PFNanoAODSchema
NanoAODSchema
ScoutingNanoAODSchema
PDUNESchema

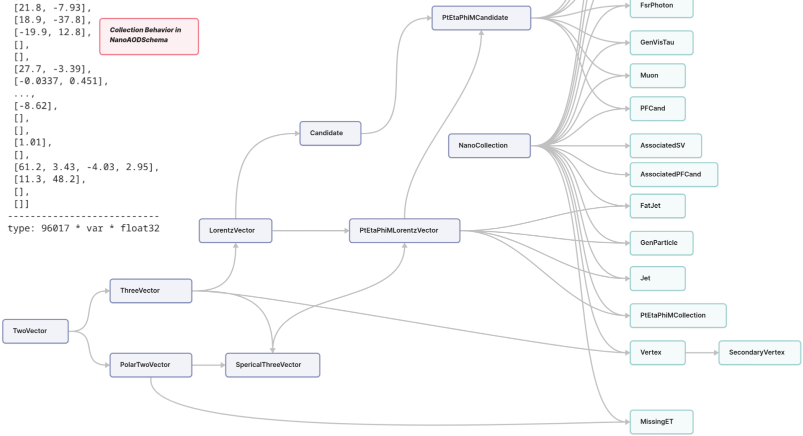
Special methods and behavior can be defined for particular schemas

```python
'px' in events.Muon.fields
```

```
False
```

```python
events.Muon.px.compute() #<--- could be called as Muon class inherits from a LorentzVector class
```

```
[[],
 [-8.55, 2.81],
 [2.57, 2.09],
 [21.8, -7.93],
 [18.9, -37.8],
 [-19.9, 12.8],
 [],
 [],
 [27.7, -3.39],
 [-0.0337, 0.451],
 ...,
 [-8.62],
 [],
 [],
 [1.01],
 [],
 [61.2, 3.43, -4.03, 2.95],
 [11.3, 48.2],
 [],
 []]
---------------------------
type: 96017 * var * float32
```



Project Goals

- Develop a **Schema for EDM4HEP** root files which can then be used to produce a schema of FCC samples that use the latest EDM4HEP format
- Develop new **Schema for FCC samples** to improve their compatibility with COFFEA
- Develop **examples of FCCAnalyses with COFFEA**

Current Status

- FCC Samples are based on the EDM4HEP root format.
- Have added schemas to support the two major EDM4Hep versions before Edm4hep v1.0 (say oldstyle) and Edm4Hep v1.0 (say newstyle),
  - The Oldstyle FCCSchema is available from COFFEA 2024.10.0
  - A pull request for the EDM4HEPSChema along with the Newstyle FCCSchema is in process

EDM4HEPSchema

A COFFEA-Schema developed for EDM4HEP v1 root files.

```
from coffea.nanoevents import NanoEventsFactory, BaseSchema, EDM4HEPSchema, FCC
```

```
edm4hep_test_file = "coffea/tests/samples/edm4hep.root"


# EDM4HEP root file with EDM4HEPSchema
edm4hep_events = NanoEventsFactory.from_root(
    edm4hep_test_file+":events",
    entry_stop=100,
    schemaclass=EDM4HEPSchema,
    delayed = False,
    uproot_options={"filter_name": lambda x: "PARAMETERS" not in x}
).events()
```

Branches saved into neat collections

```
edm4hep_events.fields
```

```
['CaloHitContributionCollection',
 'CaloHitMCParticleLinkCollection',
 'CaloHitSimCaloHitLinkCollection',
 'CalorimeterHitCollection',
 'ClusterCollection',
 'ClusterMCParticleLinkCollection',
 'EventHeader',
 'GPDoubleKeys',
 'GPDoubleValues',
 'GPFloatKeys',
 'GPFloatValues',
 'GPIntKeys',
 'GPIntValues',
 'GPStringKeys',
 'GPStringValues',
 'GeneratorEventParametersCollection',
 'GeneratorPdfInfoCollection',
 'MCParticleCollection',
 'ParticleIDCollection',
 'RawCalorimeterHitCollection',
 'RawTimeSeriesCollection',
 'RecDqdxCollection',
 'RecoMCParticleLinkCollection',
 'ReconstructedParticleCollection',
 'SimCalorimeterHitCollection',
 'SimTrackerHitCollection',
 'TimeSeriesCollection',
 'TrackCollection',
 'TrackMCParticleLinkCollection',
 'TrackerHit3DCollection',
 'TrackerHitPlaneCollection',
 'TrackerHitSimTrackerHitLinkCollection',
 'VertexCollection',
 'VertexRecoParticleLinkCollection']
```

- Branches are saved into neat collections accessible via standard Python syntax, e.g., edm4hep_events.MCParticleCollection.momentumAtEndpoint has x,y,z sub-branches
- Vector members are supported via a nested AwkwardArray
- Relations between members are created via helper functions and transforms
- The FCCSchema provides helper functions for relations in v1.0 EDM4HEP files (eg, events.EFlowPhoton.get_cluster_photons())

```
edm4hep_events.MCParticleCollection.fields
```

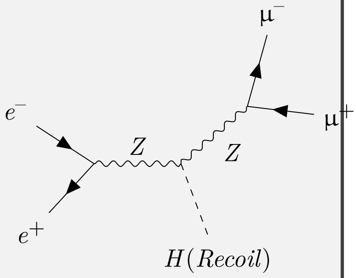
```
['PDG',
 'charge',
 'colorFlow',
 'daughters_idx_MCParticleCollection_collectionID',
 'daughters_idx_MCParticleCollection_index',
 'daughters_idx_MCParticleCollection_index_Global',
 'endpoint',
 'generatorStatus',
 'mass',
 'momentumAtEndpoint',
 'parents_idx_MCParticleCollection_collectionID',
 'parents_idx_MCParticleCollection_index',
 'parents_idx_MCParticleCollection_index_Global',
 'px',
 'py',
 'pz',
 'simulatorStatus',
 'spin',
 'time',
 'vertex']
```

ZH recoil (or mHrecoil) example

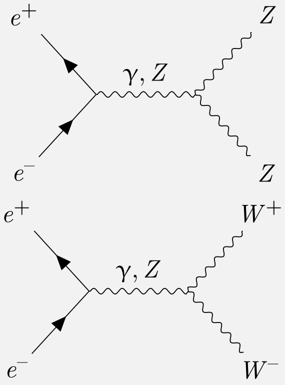
- We used the ZH Recoil example in FCCAnalysis to validate that the same results can be obtained from COFFEA as from FCCAnalysis

The Signal

$$e^+e^- \to ZH \to \mu^+\mu^- + X$$

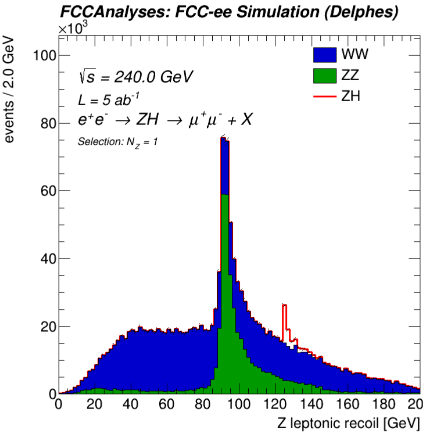


Major Backgrounds

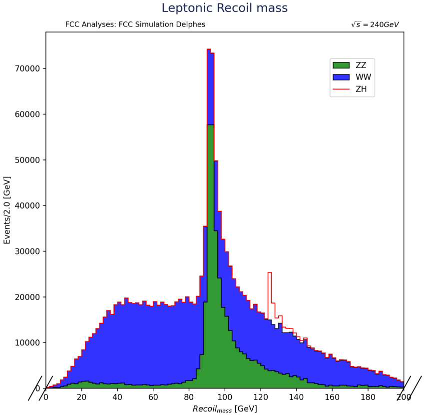


Plot: Recoil mass

Same results are obtained as desired



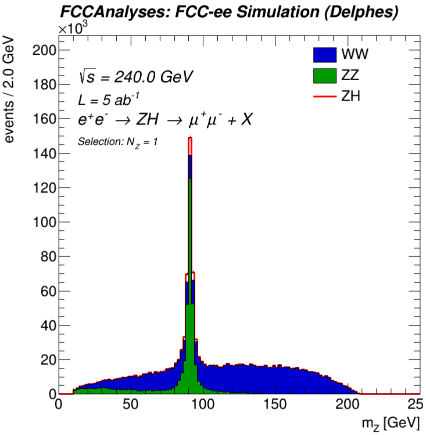
FCCAnalyses



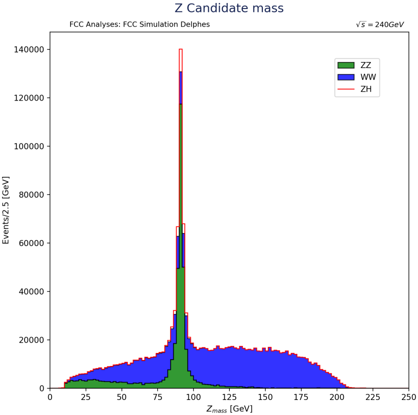
COFFEA

Plot: Z candidate mass

Same results are obtained as desired



FCCAnalyses



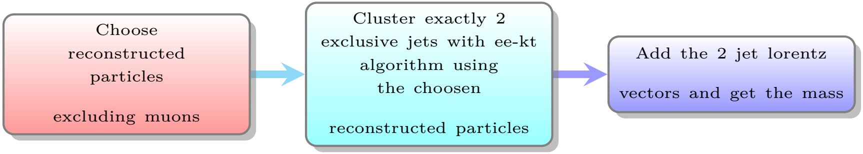
COFFEA

JetClustering example

- Utilize the python bindings for **FastJet** [2] library to cluster jets and find best the matched PDGID of the jets.

➚ Link to the example in FCCAnalyses

➚ Link to the example in coffea-fcc-analyses

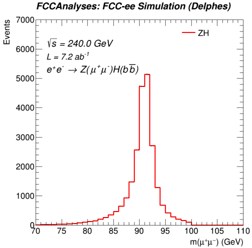


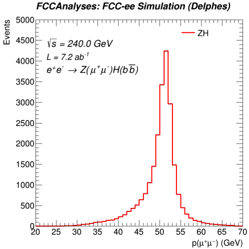
[2] https://fastjet.readthedocs.io/en/latest/

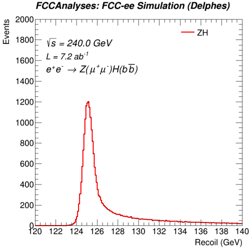
JetClustering: Z kinematics

Same results are obtained as desired

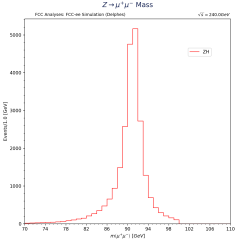
FCCAnalyses

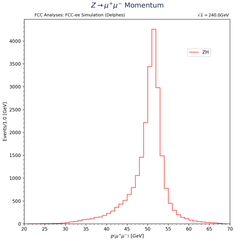


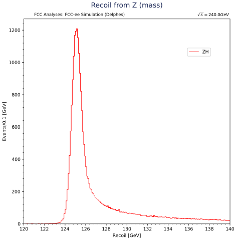




COFFEA





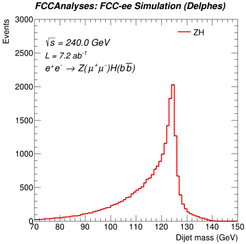


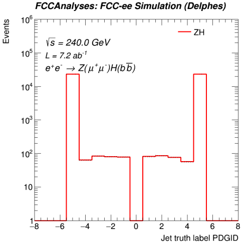
Z_{mass} Z_p $Recoil$

JetClustering: Dijet mass and Jet PDGID

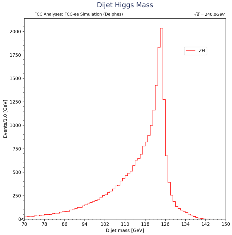
Same results are obtained as desired

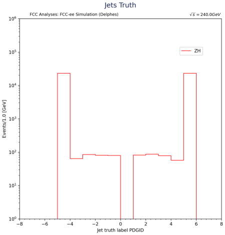
FCCAnalyses





COFFEA





$Dijet_{mass}$ $\qquad$ $Jet\ PDGID$

coffea-fcc-analyses

- Working on a repository for examples of **COFFEA based analysis** for EDM4HEP data (following the FCCAnalysis pattern). Work-in-progress examples currently available
  - **ZH recoil** using BaseSchema
  - **JetClustering** using FCCSchema and FastJet python bindings
  - **Performance comparison** using ZH recoil analysis (answer: similar performance found in both frameworks..)
  - https://github.com/prayagyadav/coffea-fcc-analyses

Next Steps

- **Automate detection of file format** and apply appropriate schema.
- **Access metadata** to more easily build relationships, helper functions, and follow future changes in edm4hep specification
- Provide **more** complete (and documented) **examples**

Thank you for your attention!

Backup Slides

About the project

HSF-India Trainee Project

Building examples for Future Circular Collider (FCC) analyses using the Columnar Framework For Effective Analysis (COFFEA) framework and developing the schema class implementation of FCC simulation samples in COFFEA [3]
July 2024 - Present

Guided by :
Dr. David Lange (Princeton University)
Dr. Bhawna Gomber (University of Hyderabad)

[3] https://research-software-collaborations.org/trainees/PrayagYadav.html

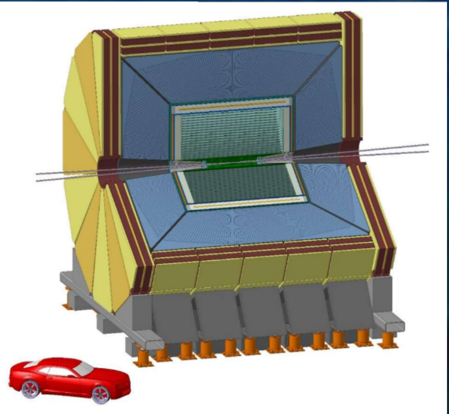
Future Circular Collider

- **Estimated cost:** Upwards of 17 Billion CHF
- **Feature:** upto 4 experiments (example: IDEA and ALLEGRO)
- **Physics outcome:** Precision measurement of Higgs and BSM physics
- **FCC Feasibility Study final results by end of 2025** [4]

[4] F. Gianotti

Future Circular Collider

- **Features:** upto 4 experiments (example: IDEA and ALLEGRO)
- **FCC Feasibility Study final results by end of 2025** [5]



IDEA concept (F. Bedeschi)

[5] F. Gianotti

FCC Analyses setup

Setup various path variables

```
source /cvmfs/sw.hsf.org/key4hep/setup.sh
```

Call fccanalysis python binary and use on your analysis file

```
fccanalysis run analysis_script.py
```

- Samples located at `/eos/experiment/fcc/*`
- fccanalysis binary located at repo FCCAnalyses/bin/
- analysis_script.py is a user defined script

The RDataFrame format

analysis_stage1.py

```
.
.
.
#Mandatory: RDFanalysis class where the use defines the
#operations on the TTree
class RDFanalysis():

#Mandatory: analyzer funtion to define the
#analyzer to process,
#please make sure you return the last dataframe,
# in this example it is df2
def analysers(df):
df2 = (
df
# define an alias for muon index collection
.Alias("Muon0", "Muon#0.index")
.
.
.
```

A look into the Schemas

Layout

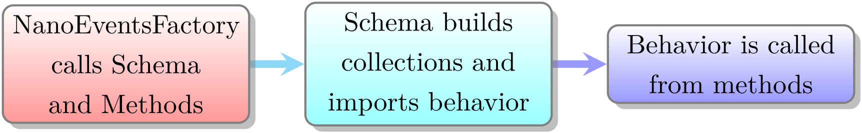
```
coffea.nanoevents/
├── factory.py     --> coffea.nanoevents.NanoEventsFactory defined here
├── __init__.py    --> imports for coffea.nanoevents
├── mapping/       --> Info on how to load samples(from_root, from_parquet etc)
├── methods/       --> Definition of mixins for various Schemas
├── schemas/       --> coffea.nanoevents.<SchemaName> defined here
├── transforms.py  --> Helper functions
└── util.py        --> Helper functions
```

What's a mixin?

A mixin class is a helper class that has no `__init__` and is merely present to add functionality to other classes. For example, Electron, Photon etc are mixin classes defined to add special functionalities to the Electron and Photon collections defined in the NanoAODSchema.

More info: `https://awkward-array.org/doc/main/reference/ak.behavior.html#mixin-decorators`

Flow of data



What's a behavior?

A behavior is a dictionary of names and mixin classes and mixin methods. Instead of defining methods to each and every class.

More info: https://awkward-array.org/doc/main/reference/ak.behavior.html

Structure of a Schema class

- Any Schema class inherits from the `BaseSchema` class.
- `BaseSchema` provides fields and their corresponding contents, through a dictionary named `_form`
- A Schema has three important methods:
  - $\rightarrow$ `__init__`, where `_build_collections(args...)` is passed and `_form` is updated
  - $\rightarrow$ `_build_collections`, that collects branches, creates LorentzVectors and collections
  - $\rightarrow$ `behavior` (classmethod) , that imports behavior from methods

Example: NanoAODSchema

/src/coffea/nanoevents/schemas/nanoaod.py

```python
class NanoAODSchema(BaseSchema):
....
def __init__(self, base_form, version="latest"):
super().__init__(base_form)
...
self._form["fields"], self._form["contents"] =
self._build_collections(
self._form["fields"], self._form["contents"]
)
...
def _build_collections(self, field_names, input_contents):
branch_forms = {k: v for k, v in zip(field_names, input_contents)}
...
@classmethod
def behavior(cls):
"""Behaviors necessary to implement this schema"""
from coffea.nanoevents.methods import nanoaod
return nanoaod.behavior
```

BaseSchema Processor

python code (1)

```
from coffea import processor
from coffea.analysis_tools import PackedSelection, Cutflow
import awkward as ak
import pandas as pd
import dask_awkward as dak
import hist.dask as hda
from collections import namedtuple
import hist
import vector
vector.register_awkward()


##########################
# Define plot properties #
##########################
plot_props = pd.DataFrame({
        'Zm':{'name':'Zm','title':'Z Candidate mass','xlabel':'$Z_{mass}$ [GeV]',
                'ylabel':'Events','bins':100,'xmin':0,'xmax':250},
        'Zm_zoom':{'name':'Zm_zoom','title':'Z Candidate mass','xlabel':'$Z_{mass}$ [GeV]',
                'ylabel':'Events','bins':40,'xmin':80,'xmax':100},
        'Recoilm':{'name':'Recoilm','title':'Leptonic Recoil mass','xlabel':'$Recoil_{mass}$ [GeV]',
                'ylabel':'Events','bins':100,'xmin':0,'xmax':200},
        'Recoilm_zoom':{'name':'Recoilm_zoom','title':'Leptonic Recoil mass','xlabel':'$Recoil_{mass}$
                [GeV]', 'ylabel':'Events','bins':200,'xmin':80,'xmax':160},
        ...
```

BaseSchema Processor

python code (2)

```
        ...
        'Recoilm_zoom2':{'name':'Recoilm_zoom2','title':'Leptonic Recoil mass','xlabel':'$Recoil_{mass}$
                [GeV]','ylabel':'Events','bins':200,'xmin':120,'xmax':140},
        'Recoilm_zoom3':{'name':'Recoilm_zoom3','title':'Leptonic Recoil mass','xlabel':'$Recoil_{mass}$
                [GeV]','ylabel':'Events','bins':400,'xmin':120,'xmax':140},
        'Recoilm_zoom4':{'name':'Recoilm_zoom4','title':'Leptonic Recoil mass','xlabel':'$Recoil_{mass}$
                [GeV]','ylabel':'Events','bins':800,'xmin':120,'xmax':140},
        'Recoilm_zoom5':{'name':'Recoilm_zoom5','title':'Leptonic Recoil mass','xlabel':'$Recoil_{mass}$
                [GeV]','ylabel':'Events','bins':2000,'xmin':120,'xmax':140},
        'Recoilm_zoom6':{'name':'Recoilm_zoom6','title':'Leptonic Recoil mass','xlabel':'$Recoil_{mass}$
                [GeV]','ylabel':'Events','bins':100,'xmin':130.3,'xmax':140}
})
def get_1Dhist(name, var, flatten=False):
'''
name: eg. Zm
var: eg. variable containing array of mass of Z
flatten: If to flatten var before fill; False by default
Returns a histogram
'''
props = plot_props[name]
if flatten : var = dak.ravel(var) # Removes None values and all the nesting
var = var[~dak.is_none(var, axis=0)] # Remove None values only
return hda.Hist.new.Reg(props.bins, props.xmin, props.xmax).Double().fill(var)
...
```

BaseSchema Processor

python code (3)

```
...
def get(events,collection,attribute,*cut):
'''Get an attribute from a branch with or without a base cut.
'''
if len(cut) != 0:
return events[collection+'/'+collection+'.'+attribute][cut[0]]
return events[collection+'/'+collection+'.'+attribute]

def get_all(events,Collection,*basecut):
'''Collect all the attributes of a collection into a namedtuple named
particle, with or without a base cut
'''
prefix = '/'.join([Collection]*2)+'.'
list_of_attr=[field.replace(prefix,'') for field in events.fields if field.startswith(prefix)]
replace_list = ['.','[',']']
valid_attr = list_of_attr
for rep in replace_list:
valid_attr = [field.replace(rep, '_') for field in valid_attr ]
part = namedtuple('particle', valid_attr)
return part(*[get(events,Collection,attr,*basecut) for attr in list_of_attr])

def get_reco(Reconstr_branch, needed_particle, events):
'''Match the Reconstructed collection to the desired particle collection.'''
part = namedtuple('particle', list(Reconstr_branch._fields))
return part(*[getattr(Reconstr_branch,attr)[get(events,needed_particle,'index')]
for attr in Reconstr_branch._fields])
```

BaseSchema Processor

python code (4)

```
...
def Reso_builder(lepton, resonance):
'''
Builds Resonance candidates
Input:    lepton(var*[var*LorentzVector]),
resonance(float)
Output: Reso([var*LorentzVecctor]) best resonance candidate in each event (maximum one per event)
'''
#Create all the combinations
combs = dak.combinations(lepton,2)
# Get dileptons
lep1 , lep2 = dak.unzip(combs)
di_lep = lep1 + lep2 # This process drops any other field except 4 momentum fields

di_lep = ak.zip({"px":di_lep.px,"py":di_lep.py,"pz":di_lep.pz,"E":di_lep.E,"q":lep1.q + lep2.q,},
with_name="Momentum4D")

# Sort by closest mass to the resonance value
sort_mask = dak.argsort(abs(resonance-di_lep.mass), axis=1)
Reso = di_lep[sort_mask]

#Choose the best candidate
#Transform the None values at axis 0 to [], so that they survive the next operation
Reso = dak.fill_none(Reso,[],axis=0)
Reso = dak.firsts(Reso) #Chooses the first elements and flattens out, [] gets converted to None
return Reso
```

BaseSchema Processor

python code (5)

```
...
##################################
#Begin the processor definition #
##################################
class mHrecoil(processor.ProcessorABC):
'''
mHrecoil example: e^+ + e^- rightarrow ZH rightarrow mu^+ mu^- + X(Recoil)
Note: Use only BaseSchema with this processor
'''
def __init__(self, ecm):
self.arg_ecm = ecm #\sqrt(s) in GeV
self.arg_zmass = 91.0 #GeV
def process(self,events):
#Create a Packed Selection object to get a cutflow later
cut = PackedSelection()
cut.add('No cut',dak.ones_like(dak.num(get(events,'ReconstructedParticles','energy'),axis=1),
dtype=bool))
# Filter out any event with no reconstructed particles and generate
Reconstructed Particle Attributes
#ak.mask preserves array length
at_least_one_recon = dak.num(get(events,'ReconstructedParticles','energy'), axis=1) > 0
good_events = dak.mask(events,at_least_one_recon)
cut.add('At least one Reco Particle', at_least_one_recon)

Reco = get_all(good_events,'ReconstructedParticles')
Muons = get_reco(Reco,'Muon#0',good_events)
```

BaseSchema Processor

python code (6)

```
...
# Create Array of Muon Lorentz Vector
Muon = ak.zip({"px":Muons.momentum_x,
        "py":Muons.momentum_y,
        "pz":Muons.momentum_z,
        "E":Muons.energy,
        "q":Muons.charge,},
with_name="Momentum4D")

# Get Muons with a pt cut , if none of the muons in an event pass the cut,
# return none, ensuring the size of the cutflow
pt_mask = dak.any(Muon.pt > 10, axis = 1)
temp = dak.mask(Muon, pt_mask)
Muon = Muon[temp.pt > 10]
cut.add('At least one Muon pt > 10', pt_mask)

# Get best Resonance around Z mass in an event
Z_cand = Reso_builder(Muon, self.arg_zmass)

# Selection 0 : Z q=0 candidate
q_mask = Z_cand.q == 0
Z_cand_sel0 = dak.mask(Z_cand, q_mask)
cut.add("Z_q = 0", q_mask)
sel0_ocl = cut.cutflow(*cut.names).yieldhist()
```

BaseSchema Processor

python code (7)

```
...
# Selection 1 : 80 < M_Z < 100
Z_mass_mask = (Z_cand.mass > 80.0) & (Z_cand.mass < 100.0)
Z_cand_sel1 = ak.mask(Z_cand, Z_mass_mask)
cut.add('80 < $M_Z$ < 100',Z_mass_mask)
sel = [*cut.names]
sel.remove(sel[-2])
sel1_ocl = cut.cutflow(*sel).yieldhist()

#Recoil Calculation
Recoil_sel0 = ak.zip({"px":0.0-Z_cand_sel0.px,"py":0.0-Z_cand_sel0.py,
        "pz":0.0-Z_cand_sel0.pz,"E":self.arg_ecm-Z_cand_sel0.E},
with_name="Momentum4D")
Recoil_sel1 = ak.zip({"px":0.0-Z_cand_sel1.px,"py":0.0-Z_cand_sel1.py,
        "pz":0.0-Z_cand_sel1.pz,"E":self.arg_ecm-Z_cand_sel1.E},
with_name="Momentum4D")

#Prepare output
#Choose the required histograms and their assigned variables to fill
names = plot_props.columns.to_list()
vars_sel0 = ([Z_cand_sel0.mass]*2) + ([Recoil_sel0.mass]*8)
vars_sel1 = ([Z_cand_sel1.mass]*2) + ([Recoil_sel1.mass]*8)
```

BaseSchema Processor

python code (8)

```
...
Output = {
        'histograms': {
                'sel0':{name:get_1Dhist(name,var) for name,var in zip(names,vars_sel0)},
                'sel1':{name:get_1Dhist(name,var) for name,var in zip(names,vars_sel1)}
        },
        'cutflow': {
                'sel0': {'Onecut':sel0_ocl[0],'Cutflow':sel0_ocl[1],'Labels':sel0_ocl[2]},
                'sel1': {'Onecut':sel1_ocl[0],'Cutflow':sel1_ocl[1],'Labels':sel1_ocl[2]}
        }
}
return Output

def postprocess(self, accumulator):
pass
```

FCCAnalyses

➚ Link to the test

Graph builder part of the histmaker code(1)

```
def build_graph(df, dataset):
results = []
df = df.Define("weight", "1.0")
weightsum = df.Sum("weight")

# select muons from Z decay and form Z/recoil mass
df = df.Alias("Muon0", "Muon#0.index")
df = df.Define("muons_all", "FCCAnalyses::ReconstructedParticle::get(Muon0, ReconstructedParticles)")
df = df.Define("muons", "FCCAnalyses::ReconstructedParticle::sel_p(25)(muons_all)")
df = df.Define("muons_p", "FCCAnalyses::ReconstructedParticle::get_p(muons)")
df = df.Define("muons_no", "FCCAnalyses::ReconstructedParticle::get_n(muons)")
df = df.Filter("muons_no >= 2")
...
```

FCCAnalyses

Graph builder part of the histmaker code(2)

```
...
df = df.Define("zmumu", "ReconstructedParticle::resonanceBuilder(91)(muons)")
df = df.Define("zmumu_m", "ReconstructedParticle::get_mass(zmumu)[0]")
df = df.Define("zmumu_p", "ReconstructedParticle::get_p(zmumu)[0]")
df = df.Define("zmumu_recoil", "ReconstructedParticle::recoilBuilder(240)(zmumu)")
df = df.Define("zmumu_recoil_m", "ReconstructedParticle::get_mass(zmumu_recoil)[0]")

# basic selection
df = df.Filter("zmumu_m > 70 && zmumu_m < 100")
df = df.Filter("zmumu_p > 20 && zmumu_p < 70")
df = df.Filter("zmumu_recoil_m < 140 && zmumu_recoil_m > 120")

# define histograms
results.append(df.Histo1D(("zmumu_m", "", *bins_m_ll), "zmumu_m"))
results.append(df.Histo1D(("zmumu_p", "", *bins_p_ll), "zmumu_p"))
results.append(df.Histo1D(("zmumu_recoil_m", "", *bins_recoil), "zmumu_recoil_m"))

return results, weightsum
```

coffea-fcc-analyses

- ➚ Link to the test

processor part of the code(1)

```python
class speed_test(processor.ProcessorABC):
'''
Processor: Define actual calculations here
'''
def __init__(self, *args, **kwargs):
pass

def process(self,events):

# Object Selections
Muons = events.ReconstructedParticles.match_collection(events.Muonidx0)
sel_muon_p_gt_25 = Muons.p > 25.0
Muons = Muons[sel_muon_p_gt_25]
Z = ReconstructedParticleUtil.resonanceBuilder(Muons, 91.0)
Recoil = ReconstructedParticleUtil.recoilBuilder(Z, 240.0)
...
```

coffea-fcc-analyses

processor part of the code(2)

```
#Event Selections
cuts = PackedSelection()
cuts.add("n_gte_2_Muons", ak.num(Muons, axis=1) >= 2 )
cuts.add("m_gt_70_Z", Z.m > 70.0 )
cuts.add("m_lt_100_Z", Z.m < 100.0 )
cuts.add("p_gt_20_Z", Z.p > 20.0 )
cuts.add("p_lt_70_Z", Z.p < 70.0 )
cuts.add("m_gt_120_Recoil", Recoil.m > 120.0 )
cuts.add("m_lt_140_Recoil", Recoil.m < 140.0 )

# Apply the event selections
Good_Z = Z[cuts.all()]
Good_Recoil = Recoil[cuts.all()]

#Prepare output
#Choose the required histograms and their assigned variables to fill
names = plot_props.columns.to_list()
vars_sel = [
Good_Recoil.m,
Good_Z.p,
Good_Z.m,
]
sel_ocl = cuts.cutflow(*cuts.names).yieldhist()
...
```

coffea-fcc-analyses

processor part of the code(3)

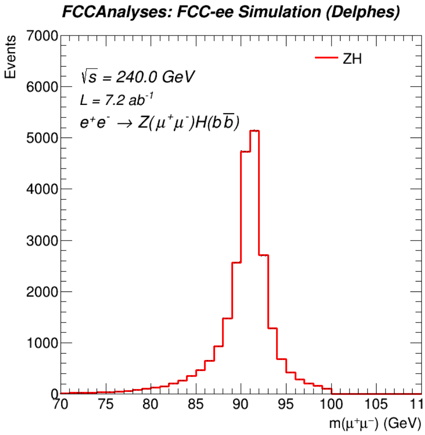
```
...
Output = {
        'histograms': {
                'sel':{
                        name:get_1Dhist(name,var,flatten=False)
                        for name,var in zip(names,vars_sel)},
        },
        'cutflow': {
                'sel': {'Onecut':sel_ocl[0],'Cutflow':sel_ocl[1],'Labels':sel_ocl[2]},
        }
}

return Output

def postprocess(self, accumulator):
pass
```

Results

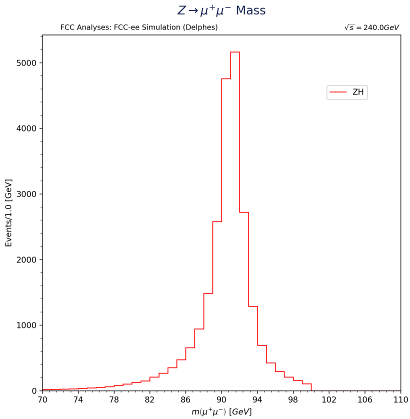
Attempt	FCCAnalyses (sec.)		coffea-fcc-analyses (sec.)	
	run	**plots**	**runner.py**	**plotter.py**
1	22.499	10.097	14.444	5.324
2	21.922	12.574	12.666	5.256
3	19.108	10.225	12.574	5.259
4	20.250	10.477	13.027	5.320
5	19.696	10.324	13.073	5.284
Mean	20.695 ± 1.30	10.238 ± 0.15	13.073 ± 0.70	5.323 ± 0.08

- **Total time taken by FCCAnalyses:** 30.933 ± 1.45 s
- **Total time taken by coffea-fcc-analyses:** 18.396 ± 0.78 s
- COFFEA is around 40 % faster (Unexpected!)

jetclustering: Z mass

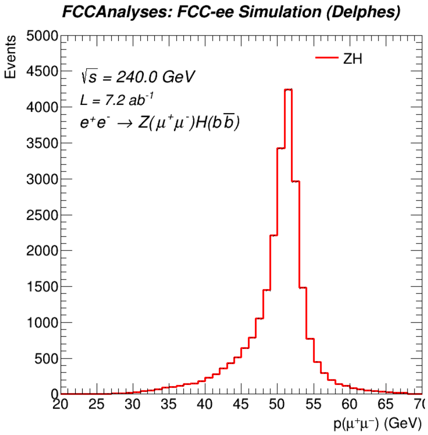


FCCAnalyses

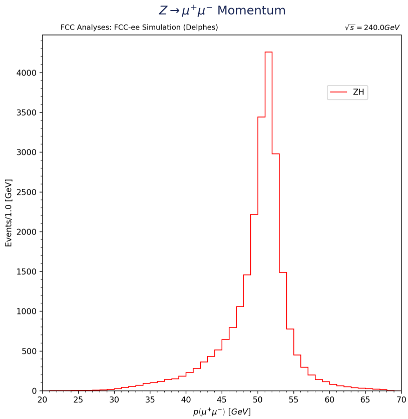


COFFEA-FCC-ANALYSES

jetclustering: Z momentum

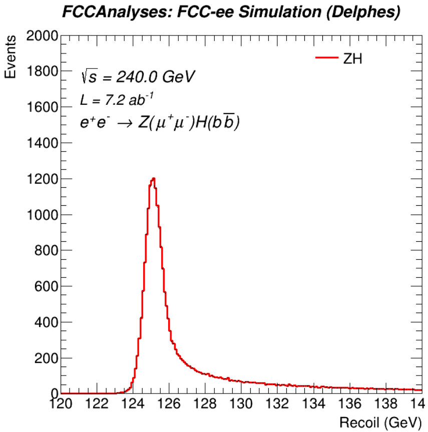


FCCAnalyses

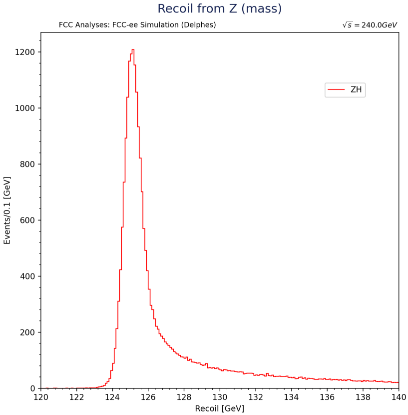


COFFEA-FCC-ANALYSES

jetclustering: Recoil from Z

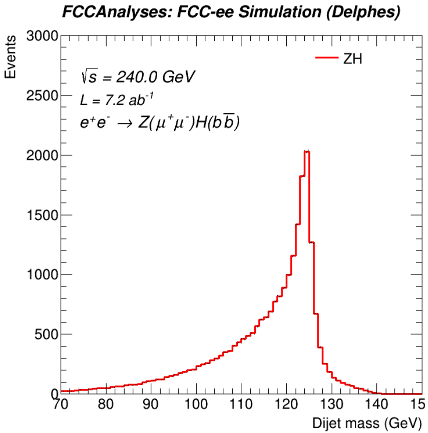


FCCAnalyses

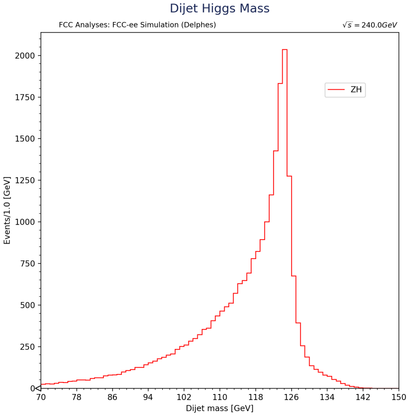


COFFEA-FCC-ANALYSES

jetclustering: Dijet Mass

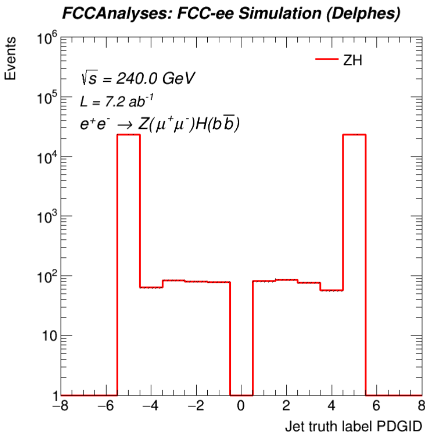


FCCAnalyses

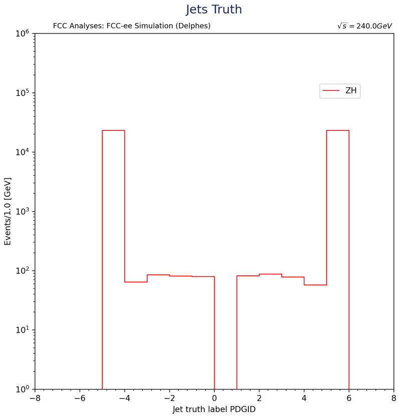


COFFEA-FCC-ANALYSES

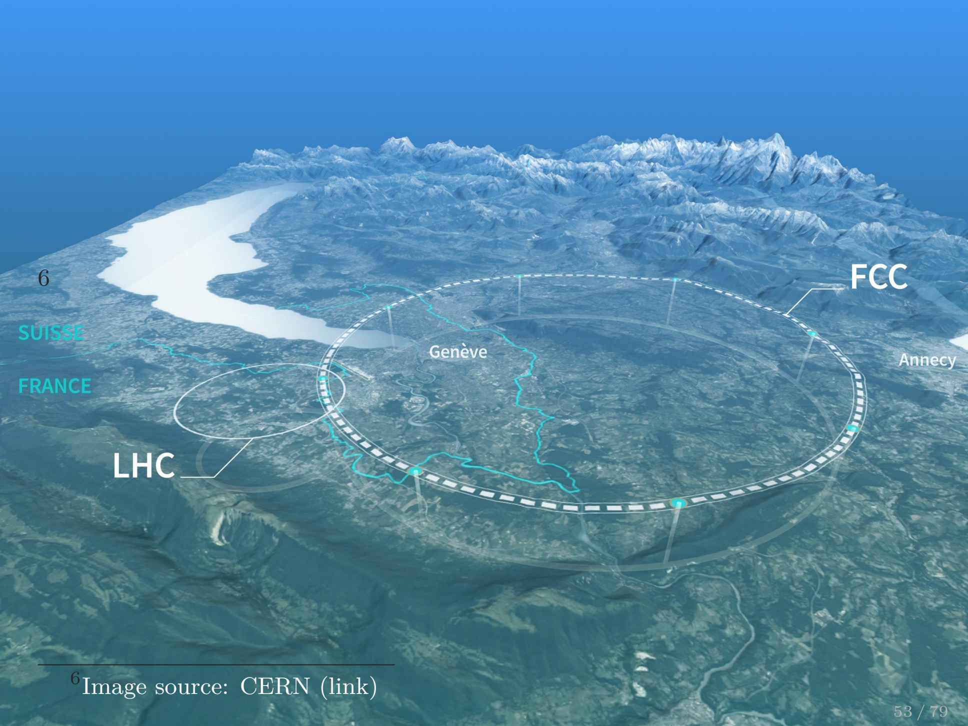
jetclustering: Truth Plot (PDGID of the best matched quark)



FCCAnalyses



COFFEA-FCC-ANALYSES



[6] Image source: CERN (link)

Future Circular Collider

- Future Circular Collider (FCC) is a possible successor to the Large Hadron Collider(LHC), spanning 90.7 km in circumference over the Swiss-French region.

Future Circular Collider

- Future Circular Collider (FCC) is a possible successor to the Large Hadron Collider(LHC), spanning 90.7 km in circumference over the Swiss-French region.
- The new tunnel is planned to initially house an electron-positron collider (FCC-ee) for 15 years, starting in the mid 2040s.

Future Circular Collider

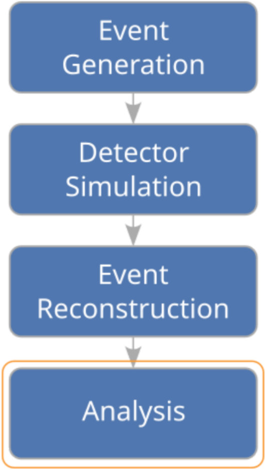
- Future Circular Collider (FCC) is a possible successor to the Large Hadron Collider(LHC), spanning 90.7 km in circumference over the Swiss-French region.
- The new tunnel is planned to initially house an electron-positron collider (FCC-ee) for 15 years, starting in the mid 2040s.
- Using the existing infrastructure, a proton-proton (and Pb ion) collider (FCC-hh) would be installed after the run of FCC-ee in 2070s for 25 years.

Future Circular Collider

- Future Circular Collider (FCC) is a possible successor to the Large Hadron Collider(LHC), spanning 90.7 km in circumference over the Swiss-French region.
- The new tunnel is planned to initially house an electron-positron collider (FCC-ee) for 15 years, starting in the mid 2040s.
- Using the existing infrastructure, a proton-proton (and Pb ion) collider (FCC-hh) would be installed after the run of FCC-ee in 2070s for 25 years.
- Expect precision measurements of Higgs properties and BSM Physics

FCCAnalyses

- **FCCAnalyses** [7] is a common framework used for FCC related analyses
- It's an **RDataFrame** framework that utilises input from the user in the form of configuration python scripts.
- It takes in "**EDM4HEP**" root files, creates an RDataFrame, performs calculation on the dataframe as defined by the user and then saves histograms.
- Find an example in upcoming slides

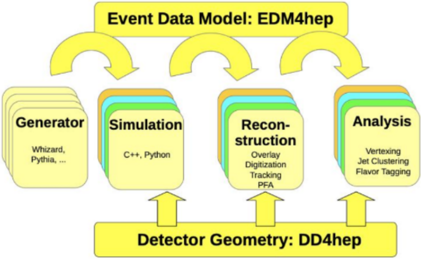


[8]

[7] FCCSW Team
[8] Juraj Smieško for FCC Week 2023

Event Data Model: EDM4HEP

- **EDM4HEP** [9] is a type of event data format that is used for FCC **event data storage** in a root file.
- Event data, generated with **podio**[10], is described by a set of standard objects, defined in a single **yaml file**.
- EDM4HEP, FCCAnalyses, podio and other useful software are shipped in the **KEY4HEP software stack**.



[11]

[9] key4hep/edm4hep
[10] key4hep/podio
[11] Juraj Smieško for FCC Week 2023

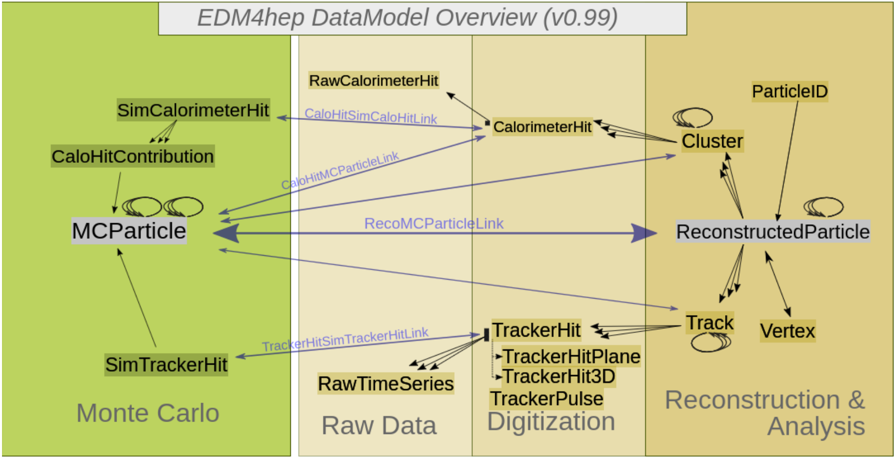
EDM4HEP yaml description

main   EDM4hep / edm4hep.yaml

Code   Blame   877 lines (801 loc) · 43.3 KB

```yaml
      "


  edm4hep::ReconstructedParticle:
    Description: "Reconstructed Particle"
    Author: "EDM4hep authors"
    Members:
      - int32_t               PDG            // PDG of the reconstructed particle.
      - float                 energy [GeV]    // energy of the reconstructed particle. Four momentum state is not kept consistent internally
      - edm4hep::Vector3f     momentum [GeV]  //  particle momentum. Four momentum state is not kept consistent internally
      - edm4hep::Vector3f     referencePoint [mm] // reference, i.e. where the particle has been measured
      - float                 charge         // charge of the reconstructed particle
      - float                 mass  [GeV]    //  mass of the reconstructed particle, set independently from four vector. Four momentum state is not kept consistent internally
      - float                 goodnessOfPID  // overall goodness of the PID on a scale of [0;1]
      - edm4hep::CovMatrix4f  covMatrix      // covariance matrix of the reconstructed particle 4vector
    OneToOneRelations:
      - edm4hep::Vertex         decayVertex    // decay vertex for the particle (if it is a composite particle)
    OneToManyRelations:
      - edm4hep::Cluster               clusters     // clusters that have been used for this particle
      - edm4hep::Track                 tracks       // tracks that have been used for this particle
      - edm4hep::ReconstructedParticle particles    // reconstructed particles that have been combined to this particle
    ExtraCode:
      includes: "#include <edm4hep/Constants.h>"
      declaration: "
      bool isCompound() const { return particles_size() > 0 ;}\n
      [[deprecated(\"use setPDG instead\")]]\n
      int32_t getType() const { return getPDG(); }\n
      /// Get the four momentum covariance matrix value for the two passed dimensions\n
      float getCovMatrix(edm4hep::FourMomCoords dimI, edm4hep::FourMomCoords dimJ) const { return getCovMatrix().getValue(dimI, dimJ); }\n
      "
    MutableExtraCode:
      includes: "#include <edm4hep/Constants.h>"
      declaration: "
      //vertex where the particle decays. This method actually returns the start vertex from the first daughter particle found.\n
```

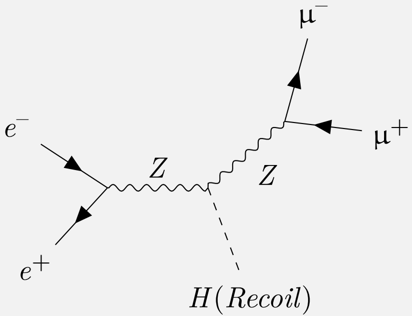
EDM4HEP relation graph



An example analysis: ZH recoil (or mHrecoil)

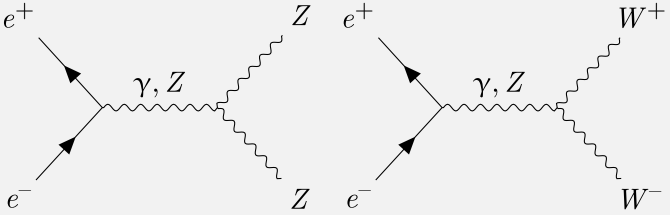
The Signal

$$e^+e^- \rightarrow ZH \rightarrow \mu^+\mu^- + X$$



An example analysis: ZH recoil (or mHrecoil)

Major Backgrounds



How is this analysis done in fccanalysis?

Setup various path variables

```
source /cvmfs/sw.hsf.org/key4hep/setup.sh
```

ZH-$\mu^+\mu^-$ recoil example

```
fccanalysis run examples/FCCee/higgs/mH-recoil/mumu/analysis_stage1.p
fccanalysis run examples/FCCee/higgs/mH-recoil/mumu/analysis_stage2.p
fccanalysis final examples/FCCee/higgs/mH-recoil/mumu/analysis_final.
fccanalysis plots examples/FCCee/higgs/mH-recoil/mumu/analysis_plots.
```

- analysis_stage1.py is for first preselection stage
- analysis_stage2.py is for the second preselection stage
- analysis_final.py is for the final selections
- analysis_plots.py is to generate plots

Selections

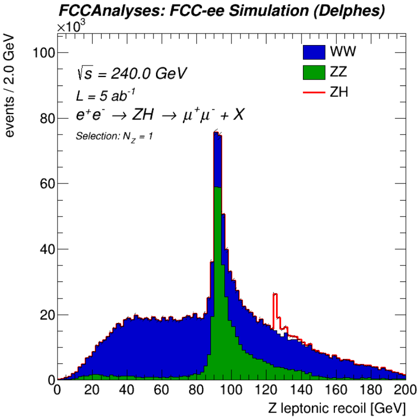
analysis_stage1.py

- Gather Muons: Match muon index with reconstructed particles
- Compute arrays (columns for RDF) to get Muon pt, eta, energy(p)
- Compute arrays of Z objects(Dimuon)(Require mass near 91 GeV)
- Compute Z p_T , η, recoil

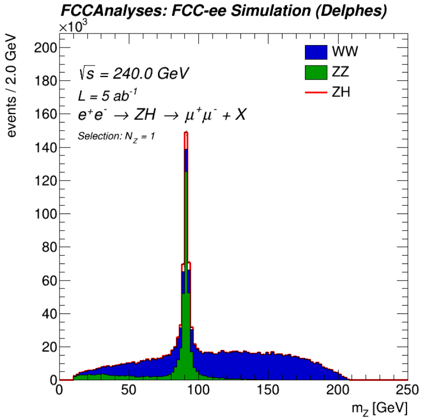
analysis_stage2.py

- Select events with exactly one Z candidate
- Get Z p_T , η, recoil mass, charge
- Compute a filtered column with $80 GeV < Z_{mass} < 100 GeV$

Plots



Recoil mass



Z candidate mass

COFFEA: In the field [1]

```
from coffea.nanoevents import NanoEventsFactory, NanoAODSchema
events = NanoEventsFactory.from_root(
    '../coffea-fcc-analyses/data/CMS_MC/AB153EDD-63CA-F340-B8E3-9A2E07FB52B3.root:Events',
    schemaclass=NanoAODSchema,
    entry_stop=100,
    metadata={'dataset':'MET'},
    delayed=False
).events()
```

```
/home/prayag/coffeafcc/development/coffea/src/coffea/nanoevents/schemas/nanoaod.py:264: RuntimeWarning: Miss
index for LowPtElectron_electronIdx => Electron
  warnings.warn(
/home/prayag/coffeafcc/development/coffea/src/coffea/nanoevents/schemas/nanoaod.py:264: RuntimeWarning: Miss
index for LowPtElectron_genPartIdx => GenPart
  warnings.warn(
/home/prayag/coffeafcc/development/coffea/src/coffea/nanoevents/schemas/nanoaod.py:264: RuntimeWarning: Miss
index for LowPtElectron_photonIdx => Photon
  warnings.warn(
```

```
events.fields
```

```
['GenJet',
 'SoftActivityJetNjets2',
 'PSWeight',
 'HLT',
 'SubJet',
 'CaloMET',
 'SoftActivityJetHT5',
 'GenMET',
 'luminosityBlock',
 'LHEReweightingWeight',
 'btagWeight',
```

COFFEA: In the field [2]

```
events.GenMET.fields
```

```
['phi', 'pt']
```

```
events.GenMET.pt
```

```
[47.7,
 45.1,
 5.03e-10,
 7.48e-09,
 9.69e-08,
 61.1,
 47.2,
 82.2,
 80.4,
 114,
 ...,
 5.37e-08,
 14.8,
 9.1e-08,
 1.06e-08,
 13.9,
 18.9,
 1.02,
 2.04,
 7.76e-07]
-------------------------------------------------
```

Structure of a Schema class

What is a collection?

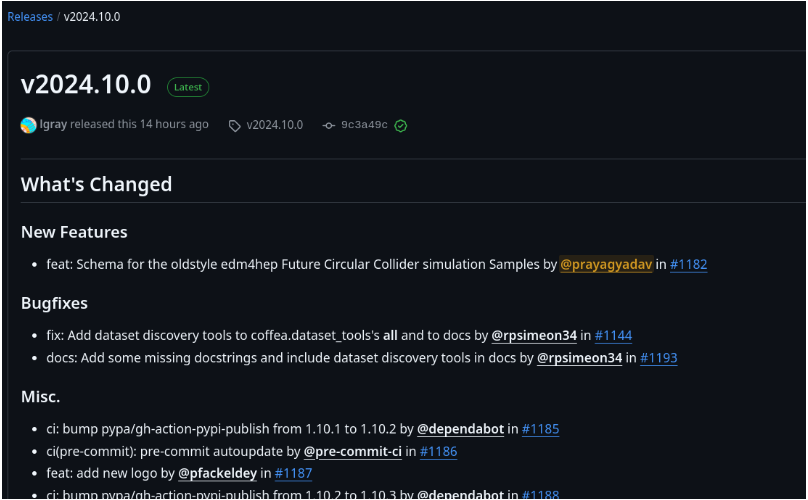
A **Collection** is a group of essentially a group branches. Eg. `ReconstructedParticle.energy`, `Reconstructed-Particle.charge`, etc can be **'zipped'** together to form the `ReconstructedParticle` collection.

A Schema has three important methods:

- → `__init__`, where `_build_collections(args...)` is called to update `_form`.
- → `_build_collections`, that collects branches, creates LorentzVectors and collections of branches
- → `behavior` (classmethod) , that imports behavior (special functionality) from methods directory

FCCSchema merged

- Oldstyle EDM4HEP FCCSchema available from COFFEA version 2024.10.0


Releases / v2024.10.0

v2024.10.0 Latest

lgray released this 14 hours ago v2024.10.0 9c3a49c

What's Changed

New Features

- feat: Schema for the oldstyle edm4hep Future Circular Collider simulation Samples by @prayagyadav in #1182

Bugfixes

- fix: Add dataset discovery tools to coffea.dataset_tools's **all** and to docs by @rpsimeon34 in #1144
- docs: Add some missing docstrings and include dataset discovery tools in docs by @rpsimeon34 in #1193

Misc.

- ci: bump pypa/gh-action-pypi-publish from 1.10.1 to 1.10.2 by @dependabot in #1185
- ci(pre-commit): pre-commit autoupdate by @pre-commit-ci in #1186
- feat: add new logo by @pfackeldey in #1187
- ci: bump pypa/gh-action-pypi-publish from 1.10.2 to 1.10.3 by @dependabot in #1188

FCCSchema in action

```
[2]: from coffea.nanoevents import NanoEventsFactory, BaseSchema, FCC
```

```
[7]: fcc = FCC.get_schema("pre-edm4hep1")
     fcc
```

```
[7]: coffea.nanoevents.schemas.fcc.FCCSchema
```

```
[11]: events = NanoEventsFactory.from_root(
          '../coffea-fcc-analyses/data/wzp6_ee_mumuH_Hbb_ecm240/events_159112833.root:events',
          schemaclass=fcc,
          entry_stop=100,
          metadata={'dataset':'ZH'},
          delayed=False,
          uproot_options={"filter_name": lambda x : "PARAMETERS" not in x}
      ).events()
```

```
/home/prayag/coffeafcc/development/coffea/src/coffea/nanoevents/mapping/uproot.py:144: UserWarning: Skipping
PARAMETERS/_intMap/_intMap.first as it is not interpretable by Uproot
  warnings.warn(f"Skipping {key} as it is not interpretable by Uproot")
/home/prayag/coffeafcc/development/coffea/src/coffea/nanoevents/mapping/uproot.py:144: UserWarning: Skipping
PARAMETERS/_intMap/_intMap.second as it is not interpretable by Uproot
  warnings.warn(f"Skipping {key} as it is not interpretable by Uproot")
/home/prayag/coffeafcc/development/coffea/src/coffea/nanoevents/mapping/uproot.py:144: UserWarning: Skipping
PARAMETERS/_floatMap/_floatMap.first as it is not interpretable by Uproot
  warnings.warn(f"Skipping {key} as it is not interpretable by Uproot")
/home/prayag/coffeafcc/development/coffea/src/coffea/nanoevents/mapping/uproot.py:144: UserWarning: Skipping
PARAMETERS/_floatMap/_floatMap.second as it is not interpretable by Uproot
  warnings.warn(f"Skipping {key} as it is not interpretable by Uproot")
/home/prayag/coffeafcc/development/coffea/src/coffea/nanoevents/mapping/uproot.py:144: UserWarning: Skipping
PARAMETERS/_stringMap/_stringMap.first as it is not interpretable by Uproot
  warnings.warn(f"Skipping {key} as it is not interpretable by Uproot")
/home/prayag/coffeafcc/development/coffea/src/coffea/nanoevents/mapping/uproot.py:144: UserWarning: Skipping
PARAMETERS/_stringMap/_stringMap.second as it is not interpretable by Uproot
```

FCCSchema in action

```
[5]: events.ReconstructedParticles.fields
```

```
[5]: ['E',
      'Electronidx0_indexGlobal',
      'MCRecoAssociationsidx0_indexGlobal',
      'Muonidx0_indexGlobal',
      'charge',
      'clusters',
      'covMatrix_10_',
      'goodnessOfPID',
      'mass',
      'particleIDs',
      'particles',
      'px',
      'py',
      'pz',
      'referencePoint',
      'tracks',
      'type']
```

```
[6]: events.Particle.fields
```

```
[6]: ['MCRecoAssociationsidx1_indexGlobal',
      'PDG',
      'charge',
      'colorFlow',
      'daughters',
      'endpoint',
      'generatorStatus',
      'mass',
      'momentumAtEndpoint',
      'parents',
      'px',
      'py',
      'pz',
```

```
[7]: events.Particle.get_daughters.PDG
```

```
[7]: [[[11, -11], [11, -11], [11], [-11], [13, -13, 25], ..., [], [], [], [], []],
      [[11, -11], [11, -11], [11], [-11], [13, -13, 25], ..., [], [], [], [], []],
      [[11, -11], [11, -11], [11], [-11], [13, -13, 25], ..., [], [], [], [], []],
      [[11, -11], [11, -11], [11], [-11], [13, ..., 25], ..., [], [22, 22], [], []],
      [[11, -11], [11, -11], [11], [-11], [13, -13, 25], ..., [], [], [], [], []],
      [[11, -11], [11, -11], [11], [-11], [13, -13, 25], ..., [...], [], [], [], []],
      [[11, -11], [11, -11], [11], [-11], [13, -13, 25], ..., [...], [], [], [], []],
      [[11, -11], [11, -11], [11], [-11], [13, -13, 25], ..., [], [], [], [], []],
      [[11, -11], [11, -11], [11], [-11], [13, -13, 25], ..., [], [], [], [], []],
      [[11, -11], [11, -11], [11], [-11], [13, -13, 25], ..., [], [], [], [], []],
      ...,
      [[11, -11], [11, -11], [11], [-11], [13, -13, 25], ..., [...], [], [], [], []],
      [[11, -11], [11, -11], [11], [-11], [13, -13, 25], ..., [...], [], [], [], []],
      [[11, -11], [11, -11], [11], [-11], [13, -13, 25], ..., [], [], [], [], []],
      [[11, -11], [11, -11], [11], [-11], [13, -13, 25], ..., [], [], [], [], []],
      [[11, -11], [11, -11], [11], [-11], [13, -13, 25], ..., [], [], [], [], []],
      [[11, -11], [11, -11], [11], [-11], [13, -13, 25], ..., [], [], [], [], []],
      [[11, -11], [11, -11], [11], [-11], [13, -13, 25], ..., [...], [], [], [], []],
      [[11, -11], [11, -11], [11], [-11], [13, -13, 25], ..., [], [], [], [], []],
      [[11, -11], [11, -11], [11], [-11], [13, -13, 25], ..., [], [], [], [], []]]
     -------------------------------------------------------------------------------
     type: 100 * var * var * ?int32[parameters={"__doc__": "PDG[Particle_]"}]
```

A simple speed test

- With a simple example, lets compare the speed of FCCAnalyses and coffea-fcc-analyses
- The simple analysis will plot Z boson mass and Recoil from it, in the ZH sample. The Z boson is assumed to decay to two muons.
- FCCAnalyses works after sourcing the key4hep stack.
- COFFEA works after calling the container for the same.
- For a fair evaluation, lets time both of them, after the stack or container is loaded (as it needs to be done only once per session)

A simple speed test

- With a simple example, lets compare the speed of FCCAnalyses and coffea-fcc-analyses
- The simple analysis will plot Z boson mass and Recoil from it, in the ZH sample. The Z boson is assumed to decay to two muons.
- FCCAnalyses works after sourcing the key4hep stack.
- COFFEA works after calling the container for the same.
- For a fair evaluation, lets time both of them, after the stack or container is loaded (as it needs to be done only once per session)

A simple speed test

- With a simple example, lets compare the speed of FCCAnalyses and coffea-fcc-analyses
- The simple analysis will plot Z boson mass and Recoil from it, in the ZH sample. The Z boson is assumed to decay to two muons.
- FCCAnalyses works after sourcing the key4hep stack.
- COFFEA works after calling the container for the same.
- For a fair evaluation, lets time both of them, after the stack or container is loaded (as it needs to be done only once per session)

A simple speed test

- With a simple example, lets compare the speed of FCCAnalyses and coffea-fcc-analyses
- The simple analysis will plot Z boson mass and Recoil from it, in the ZH sample. The Z boson is assumed to decay to two muons.
- FCCAnalyses works after sourcing the key4hep stack.
- COFFEA works after calling the container for the same.
- For a fair evaluation, lets time both of them, after the stack or container is loaded (as it needs to be done only once per session)

A simple speed test

- With a simple example, lets compare the speed of FCCAnalyses and coffea-fcc-analyses
- The simple analysis will plot Z boson mass and Recoil from it, in the ZH sample. The Z boson is assumed to decay to two muons.
- FCCAnalyses works after sourcing the key4hep stack.
- COFFEA works after calling the container for the same.
- For a fair evaluation, lets time both of them, after the stack or container is loaded (as it needs to be done only once per session)

Speed Test

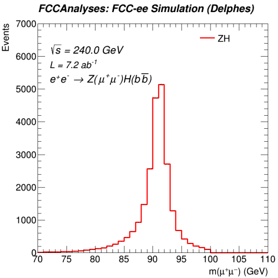
FCCAnalyses

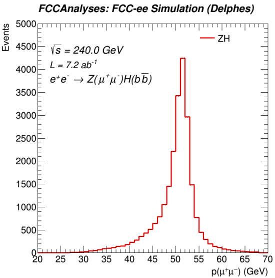
```
time fccanalyses run histmaker.py
time fccanalyses plots plots.py
```

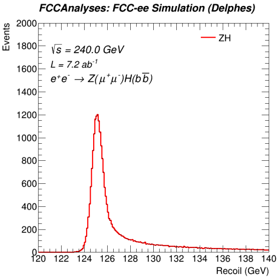
coffea-fcc-analyses

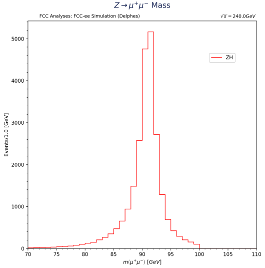
```
time python runner.py -e dask
time python plotter.py
```

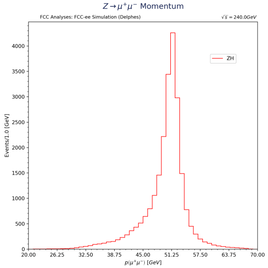
Results

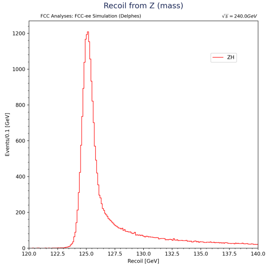












Z_{mass} Z_p $Recoil$

Results

Attempt	**FCCAnalyses (sec.)**		**coffea-fcc-analyses (sec.)**	
	run	**plots**	**runner.py**	**plotter.py**
1	22.5	10.1	14.4	5.3
2	21.9	12.6	12.7	5.3
3	19.1	10.2	12.6	5.3
4	20.2	10.5	13.0	5.3
5	19.7	10.3	13.1	5.3
Mean	20.7 ± 1.3	10.2 ± 0.2	13.1 ± 0.7	5.3 ± 0.1

- **Total time taken by FCCAnalyses:** 30.9 ± 1.5 s
- **Total time taken by coffea-fcc-analyses:** 18.4 ± 0.8 s
- Coffea with FCCSchema is around 40 % faster on this simple example. Can conclude that there is no performance problem with our Coffea implementation

Support for vector members

Vector Members are supported out of the box, with direct branches saved for easy access.

```yaml
edm4hep::ParticleID:
  Description:  "ParticleID"
  Author: "EDM4hep authors"
  Members:
    - int32_t   type           // userdefined type
    - int32_t   PDG            // PDG code of this id - ( 999999 ) if unknown
    - int32_t   algorithmType  // type of the algorithm/module that created this hypothesis
    - float likelihood     // likelihood of this hypothesis - in a user defined normalization
  VectorMembers:
    - float parameters     // parameters associated with this hypothesis
  OneToOneRelations:
    - edm4hep::ReconstructedParticle particle // the particle from which this PID has been computed
```

```python
parameters = edm4hep_events.ParticleIDCollection.parameters
parameters
```

```
[[[46, 47, 48, 49, 50]],
 [[46, 47, 48, 49, 50]],
 [[46, 47, 48, 49, 50]]]
-----------------------------------------------------------------------------------------------
type: 3 * var * [var * float64, parameters={"__doc__": "parameters associated with this hypothesis"}]
```

Support for Relations : One2One and One2Many

Relation branches could be printed and Mapped to their targets.

```
edm4hep::ReconstructedParticle:
  Description: "Reconstructed Particle"
  Author: "EDM4hep authors"
  Members:
    - int32_t              PDG            // PDG of the reconstructed particle.
    - float                energy [GeV]    // energy of the reconstructed particle. Four momentum state is not kept c
    - edm4hep::Vector3f    momentum [GeV]  //  particle momentum. Four momentum state is not kept consistent internal
    - edm4hep::Vector3f    referencePoint [mm] // reference, i.e. where the particle has been measured
    - float                charge         // charge of the reconstructed particle
    - float                mass  [GeV]    //  mass of the reconstructed particle, set independently from four vector.
consistent internally
    - float                goodnessOfPID  // overall goodness of the PID on a scale of [0;1]
    - edm4hep::CovMatrix4f covMatrix      // covariance matrix of the reconstructed particle 4vector
  OneToOneRelations:
    - edm4hep::Vertex        decayVertex    // decay vertex for the particle (if it is a composite particle)
  OneToManyRelations:
    - edm4hep::Cluster               clusters     // clusters that have been used for this particle
    - edm4hep::Track                 tracks       // tracks that have been used for this particle
    - edm4hep::ReconstructedParticle particles    // reconstructed particles that have been combined to this particle
```

```
edm4hep_events.ReconstructedParticleCollection.List_Relations
```

```
{'clusters_idx_ClusterCollection_collectionID',
 'clusters_idx_ClusterCollection_index',
 'clusters_idx_ClusterCollection_index_Global',
 'decayVertex_idx_VertexCollection_collectionID',
 'decayVertex_idx_VertexCollection_index',
 'decayVertex_idx_VertexCollection_index_Global',
 'particles_idx_ReconstructedParticleCollection_collectionID',
 'particles_idx_ReconstructedParticleCollection_index',
 'particles_idx_ReconstructedParticleCollection_index_Global',
 'tracks_idx_TrackCollection_collectionID',
 'tracks_idx_TrackCollection_index',
 'tracks_idx_TrackCollection_index_Global'}
```

```
edm4hep_events.ReconstructedParticleCollection.Map_Relation(generic_name='decayVertex', target_name='VertexCollection')
```

```
[[{algorithmType: 48, chi2: 43, covMatrix: {...}, ndf: 44, ...}],
 [{algorithmType: 48, chi2: 43, covMatrix: {...}, ndf: 44, ...}],
 [{algorithmType: 48, chi2: 43, covMatrix: {...}, ndf: 44, ...}]]
------------------------------------------------------------------------------------------------------
```

```
edm4hep_events.ReconstructedParticleCollection.Map_Relation('tracks','TrackCollection')
```

```
[[[{Nholes: 105, chi2: 43, ndf: 44, subdetectorHitNumbers: [...], ...}]],
 [[{Nholes: 105, chi2: 43, ndf: 44, subdetectorHitNumbers: [...], ...}]],
 [[{Nholes: 105, chi2: 43, ndf: 44, subdetectorHitNumbers: [...], ...}]]]
------------------------------------------------------------------------------
```

FCCSchema

- Pregenerated samples from datasets: **Spring2021** and **Winter2023** have the oldstyle edm4hep root structure.
- Oldstyle FCCSchema is a stand-alone schema built to interpret the pregenerated samples.
- Newstyle FCCSchema builds on top of EDM4HEPSchema to provide wrapper methods and extra syntax for a FCC specialized use.

```
# events.ParticleIDs.get_reconstructedparticles

# events.EFlowPhoton.get_cluster_photons
# events.EFlowPhoton.get_hits

# events.EFlowTrack.get_tracks
# events.EFlowTrack.get_trackerhits

# events.Particle.get_parents
# events.Particle.get_daughters

# events.ReconstructedParticles.List_Links <--- List the available links
# events.ReconstructedParticles.match_gen <--- match reco particle with their MC counterparts
```

BaseSchema Processor

12

python code (1)

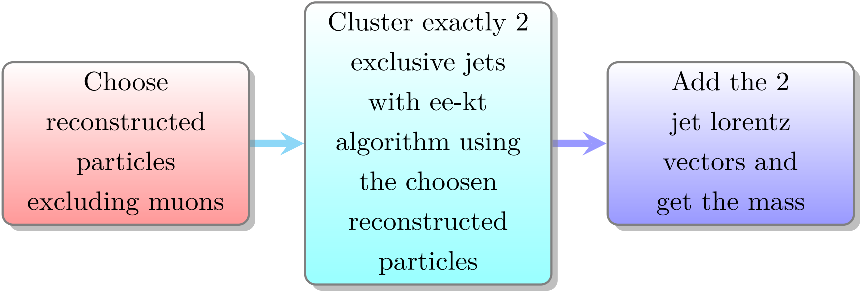
```python
from coffea import processor
from coffea.analysis_tools import PackedSelection, Cutflow
import awkward as ak
import pandas as pd
import dask_awkward as dak
import hist.dask as hda
from collections import namedtuple
import hist
import vector
vector.register_awkward()

##########################
# Define plot properties #
##########################
plot_props = pd.DataFrame({
        'Zm':{'name':'Zm','title':'Z Candidate mass','xlabel':'$Z_{mass}$ [GeV]',
                'ylabel':'Events','bins':100,'xmin':0,'xmax':250},
        'Zm_zoom':{'name':'Zm_zoom','title':'Z Candidate mass','xlabel':'$Z_{mass}$ [GeV]',
                'ylabel':'Events','bins':40,'xmin':80,'xmax':100},
        'Recoilm':{'name':'Recoilm','title':'Leptonic Recoil mass','xlabel':'$Recoil_{mass}$ [GeV]',
                'ylabel':'Events','bins':100,'xmin':0,'xmax':200},
        'Recoilm_zoom':{'name':'Recoilm_zoom','title':'Leptonic Recoil mass','xlabel':'$Recoil_{mass}$
                [GeV]', 'ylabel':'Events','bins':200,'xmin':80,'xmax':160},
        'Recoilm_zoom2':{'name':'Recoilm_zoom2','title':'Leptonic Recoil mass','xlabel':'$Recoil_{mass}$
                [GeV]','ylabel':'Events','bins':200,'xmin':120,'xmax':140},
        'Recoilm_zoom3':{'name':'Recoilm_zoom3','title':'Leptonic Recoil mass','xlabel':'$Recoil_{mass}$
                [GeV]','ylabel':'Events','bins':400,'xmin':120,'xmax':140},
        'Recoilm_zoom4':{'name':'Recoilm_zoom4','title':'Leptonic Recoil mass','xlabel':'$Recoil_{mass}$
                [GeV]','ylabel':'Events','bins':800,'xmin':120,'xmax':140},
        'Recoilm_zoom5':{'name':'Recoilm_zoom5','title':'Leptonic Recoil mass','xlabel':'$Recoil_{mass}$
                [GeV]','ylabel':'Events','bins':2000,'xmin':120,'xmax':140},
        'Recoilm_zoom6':{'name':'Recoilm_zoom6','title':'Leptonic Recoil mass','xlabel':'$Recoil_{mass}$
                [GeV]','ylabel':'Events','bins':100,'xmin':130.3,'xmax':140}
})
def get_1Dhist(name, var, flatten=False):
'''
name: eg. Zm
var: eg. variable containing array of mass of Z
flatten: If to flatten var before fill; False by default
Returns a histogram
'''
props = plot_props[name]
if flatten : var = dak.ravel(var) # Removes None values and all the nesting
var = var[~dak.is_none(var, axis=0)] # Remove None values only
return hda.Hist.new.Reg(props.bins, props.xmin, props.xmax).Double().fill(var)
def get(events,collection,attribute,*cut):
'''Get an attribute from a branch with or without a base cut.
'''
if len(cut) != 0:
return events[collection+'/'+collection+'.'+attribute][cut[0]]
return events[collection+'/'+collection+'.'+attribute]
def get_all(events,Collection,*basecut):
'''Collect all the attributes of a collection into a namedtuple named
particle, with or without a base cut
'''
prefix = '/'.join([Collection]*2)+'.'
list_of_attr=[field.replace(prefix,'') for field in events.fields if field.startswith(prefix)]
replace_list = ['.','[',']']
valid_attr = list_of_attr
for rep in replace_list:
valid_attr = [field.replace(rep, '_') for field in valid_attr ]
part = namedtuple('particle', valid_attr)
return part(*[get(events,Collection,attr,*basecut) for attr in list_of_attr])
def get_reco(Reconstr_branch, needed_particle, events):
'''Match the Reconstructed collection to the desired particle collection.'''
part = namedtuple('particle', list(Reconstr_branch._fields))
return part(*[getattr(Reconstr_branch,attr)[get(events,needed_particle,'index')]
for attr in Reconstr_branch._fields])
def Reso_builder(lepton, resonance):
'''
Builds Resonance candidates
Input:    lepton(var*[var*LorentzVector]),
resonance(float)
Output: Reso([var*LorentzVecctor]) best resonance candidate in each event (maximum one per event)
'''
#Create all the combinations
combs = dak.combinations(lepton,2)
# Get dileptons
lep1 , lep2 = dak.unzip(combs)
di_lep = lep1 + lep2 # This process drops any other field except 4 momentum fields
di_lep = ak.zip({"px":di_lep.px,"py":di_lep.py,"pz":di_lep.pz,"E":di_lep.E,"q":lep1.q + lep2.q,},
with_name="Momentum4D")
# Sort by closest mass to the resonance value
sort_mask = dak.argsort(abs(resonance-di_lep.mass), axis=1)
Reso = di_lep[sort_mask]
```

python code (2)

```python
#Choose the best candidate
#Transform the None values at axis 0 to [], so that they survive the next operation
Reso = dak.fill_none(Reso,[],axis=0)
Reso = dak.firsts(Reso) #Chooses the first elements and flattens out, [] gets converted to None
return Reso
'''
#################################
#Begin the processor definition #
#################################
class mHrecoil(processor.ProcessorABC):
'''
mHrecoil example: e^+ + e^- rightarrow ZH rightarrow mu^+ mu^- + X(Recoil)
Note: Use only BaseSchema with this processor
'''
def __init__(self, ecm):
self.arg_ecm = ecm #\sqrt(s) in GeV
self.arg_zmass = 91.0 #GeV
def process(self,events):
#Create a Packed Selection object to get a cutflow later
cut = PackedSelection()
cut.add('No cut',dak.ones_like(dak.num(get(events,'ReconstructedParticles','energy'),axis=1),
dtype=bool))
# Filter out any event with no reconstructed particles and generate
Reconstructed Particle Attributes
#ak.mask preserves array length
at_least_one_recon = dak.num(get(events,'ReconstructedParticles','energy'), axis=1) > 0
good_events = dak.mask(events,at_least_one_recon)
cut.add('At least one Reco Particle', at_least_one_recon)
Reco = get_all(good_events,'ReconstructedParticles')
Muons = get_reco(Reco,'Muon#0',good_events)
# Create Array of Muon Lorentz Vector
Muon = ak.zip({"px":Muons.momentum_x,
        "py":Muons.momentum_y,
        "pz":Muons.momentum_z,
        "E":Muons.energy,
        "q":Muons.charge,},
with_name="Momentum4D")
# Get Muons with a pt cut , if none of the muons in an event pass the cut,
# return none, ensuring the size of the cutflow
pt_mask = dak.any(Muon.pt > 10, axis = 1)
temp = dak.mask(Muon, pt_mask)
Muon = Muon[temp.pt > 10]
cut.add('At least one Muon pt > 10', pt_mask)
# Get best Resonance around Z mass in an event
Z_cand = Reso_builder(Muon, self.arg_zmass)
# Selection 0 : Z q=0 candidate
q_mask = Z_cand.q == 0
Z_cand_sel0 = dak.mask(Z_cand, q_mask)
cut.add("Z_q = 0", q_mask)
sel0_ocl = cut.cutflow(*cut.names).yieldhist()
# Selection 1 : 80 < M_Z < 100
Z_mass_mask = (Z_cand.mass > 80.0) & (Z_cand.mass < 100.0)
Z_cand_sel1 = ak.mask(Z_cand, Z_mass_mask)
cut.add('80 < $M_Z$ < 100',Z_mass_mask)
sel = [*cut.names]
sel.remove(sel[-2])
sel1_ocl = cut.cutflow(*sel).yieldhist()
#Recoil Calculation
Recoil_sel0 = ak.zip({"px":0.0-Z_cand_sel0.px,"py":0.0-Z_cand_sel0.py,
        "pz":0.0-Z_cand_sel0.pz,"E":self.arg_ecm-Z_cand_sel0.E},
with_name="Momentum4D")
Recoil_sel1 = ak.zip({"px":0.0-Z_cand_sel1.px,"py":0.0-Z_cand_sel1.py,
        "pz":0.0-Z_cand_sel1.pz,"E":self.arg_ecm-Z_cand_sel1.E},
with_name="Momentum4D")
#Prepare output
#Choose the required histograms and their assigned variables to fill
names = plot_props.columns.to_list()
vars_sel0 = ([Z_cand_sel0.mass]*2) + ([Recoil_sel0.mass]*8)
vars_sel1 = ([Z_cand_sel1.mass]*2) + ([Recoil_sel1.mass]*8)
Output = {
        'histograms': {
                'sel0':{name:get_1Dhist(name,var) for name,var in zip(names,vars_sel0)},
                'sel1':{name:get_1Dhist(name,var) for name,var in zip(names,vars_sel1)}
        },
        'cutflow': {
                'sel0': {'Onecut':sel0_ocl[0],'Cutflow':sel0_ocl[1],'Labels':sel0_ocl[2]},
                'sel1': {'Onecut':sel1_ocl[0],'Cutflow':sel1_ocl[1],'Labels':sel1_ocl[2]}
        }
}
return Output
def postprocess(self, accumulator):
pass
```

[12] Full, visible code in backup slides

Main stages of the analysis



End of the presentation