# FCCAnalyses (for the cases-studies but not only)

Clement Helsens CERN-EP

# Outline

- Introduction
- Overall structure
- Workflow
- Examples
  - Higgs Recoil
  - Flavour physics at Z pole

# Introduction

FCCAnalyses: https://github.com/HEP-FCC/FCCAnalyses/

- Originally developed for FCC-hh physics studies
  - Was based on heppy (full python analysis framework from CMS)
    - There is a heppy-legacy branch for nostalgic people
  - Was used to produced FCC-hh physics analyses published in the CDR

- Transitioned to RDataFrame
  - Heppy was slow
  - Achieved much higher performances
  - Edm4hep aware

# Common software?

- FCCAnalyses
  - Is a common tool for analyzing large amount of data using RootDataFrame and produce flat ntuple
  - Is composed of a library of C++ analysers and python configurations
    - C++ analysers are developed in common
    - Python code specific to the analysis to define the analysers, output variables, input samples
  - Use those flat ntuples
    - To produce final variables for analysis, for MVA training, and for plotting
    - To run decay selector for flavour physics
  - Is a-priori transparent to the input format

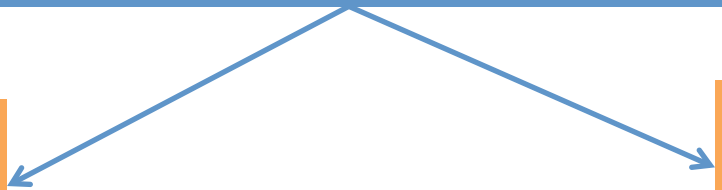
- EDM4Hep
  - EDM4Hep is the common event data model
  - Is the only supported input format at the moment but this can evolve
  - Will be used for the very long term for any simulation (so better to start using it now!)

# Overall structure - 1

- Set of tools to help processing the output of ‘simulation’
  - Agnostic to the type of simulation but specific reader functions are required
  - Build a common set of utility functions, algorithms for common use
  - Still possible for users to test their algorithms locally before publishing them

- How is FCCAnalyses structured up to now (all in the same repository)

Analysis configuration
4 python scripts to configure:
1. Samples to run over (common production, EDM4Hep)
2. Functions/algorithm to call to produce variables of interests in a flat ntuple
3. Event selection and histograms definition using the flat ntuples
4. Plotting configuration



Common utility functions, algorithm, etc…
**C++ library**

Common interface code
Sample database, RdataFrame, plotting
**Python**

# Overall structure - 2

- How is FCCAnalyses will be structured for the **case studies** (two separate repositories)

Analysis configuration
4 python scripts to configure:
1. Samples to run over (common production, EDM4Hep)
2. Functions/algorithm to call to produce variables of interests in a flat ntuple
3. Event selection and histograms definition using the flat ntuples
4. Plotting configuration

https://github.com/HEP-FCC/HEP-FCC/FCCeePhysicsPerformance/
Analysis configuration part close to the case studies and analysis specific code (like fitting, specific calculations, etc…)

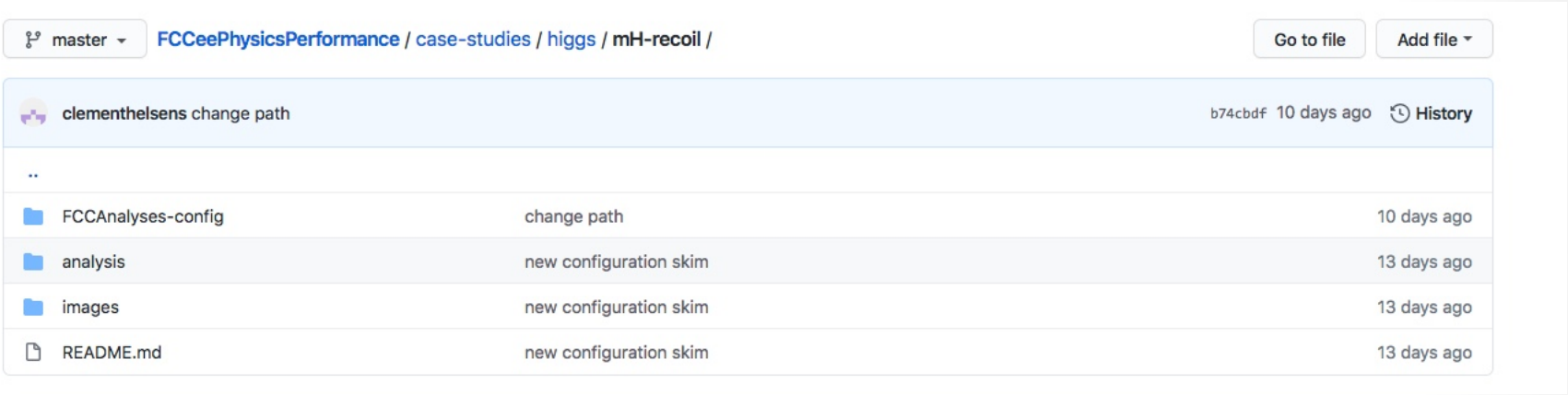
Common utility functions, algorithm, etc…
**C++ library**

Common interface code
Sample database, RdataFrame, plotting
**Python**

https://github.com/HEP-FCC/FCCAnalyses/
Centrally installed on cvmfs and accessible by common setup. But totally OK to use private version.

# Overall structure - 3

- How is FCCAnalyses will be structured for the **case studies** (two separate repositories)

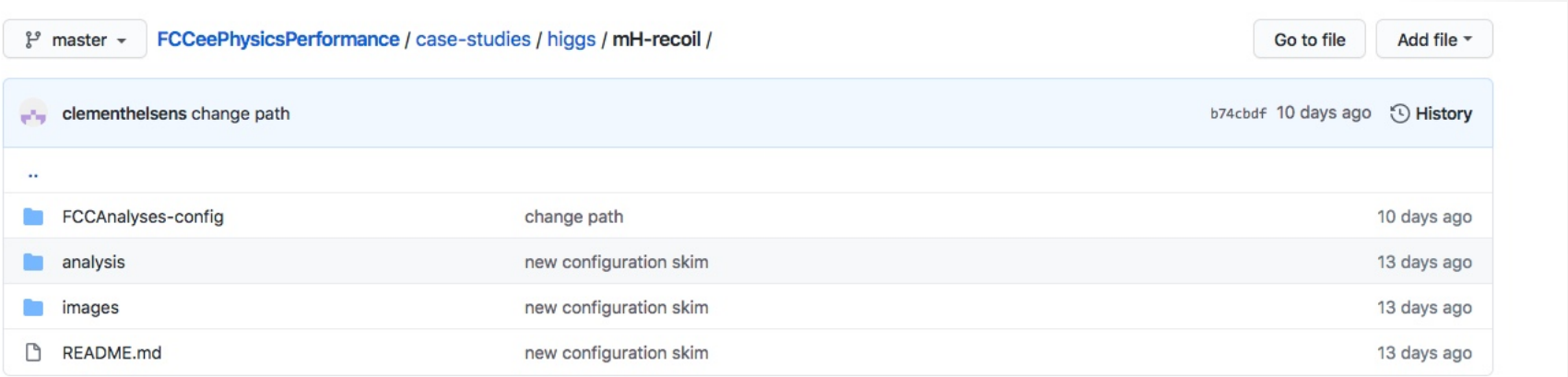



- Each case study highly encouraged to have it's own FCCAnalyses-config

Analysis configuration
4 python scripts to configure:
1. Samples to run over (common production, EDM4Hep)
2. Functions/algorithm to call to produce variables of interests in a flat ntuple
3. Event selection and histograms definition using the flat ntuples
4. Plotting configuration

# Overall structure - 4

- How is FCCAnalyses will be structured for the **case studies** (two separate repositories)




- But still possible to build and use custom algorithm provided they are properly linked

Common and user specific utility functions, algorithm, etc…
C++ library

Home | About | Contact | 100TeV FCC Physics | 27TeV HE-LHC Physics | Full Simulation FCChh | FCCee Physics | Stat

**Delphes FCCee Physic events tmp (IDEA with Track Covariance)**

240

| NO | NAME | NEVENTS | NWEIGHTS | NFILES | NBAD | NEOS | SIZE (GB) | OUTPUT PATH | MAIN PROCESS | FINAL STATES | CROSS SECTION (PB) | K-FACTOR | MATCHING EFF |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 35 | p8_ee_ZH_ecm240 | 9,920,000 | 0 | 992 | 0 | 992 | 90.31 | /eos/experiment/fcc/ee/generation/DelphesEvents/fcc_tmp /p8_ee_ZH_ecm240/ | ZH ecm=240GeV | inclusive decays | 0.201037 | 1.0 | 1.0 |
| 36 | p8_ee_ZZ_ecm240 | 9,980,000 | 0 | 998 | 0 | 998 | 67.76 | /eos/experiment/fcc/ee/generation/DelphesEvents/fcc_tmp /p8_ee_ZZ_ecm240/ | ZZ ecm=240GeV | inclusive decays | 1.35899 | 1.0 | 1.0 |
| 37 | p8_ee_WW_ecm240 | 10,000,000 | 0 | 1000 | 0 | 1000 | 65.44 | /eos/experiment/fcc/ee/generation/DelphesEvents/fcc_tmp /p8_ee_WW_ecm240/ | WW ecm=240GeV | inclusive decays | 16.4385 | 1.0 | 1.0 |

Common samples are available on eos and can be accessed through a database

But possible to by pass this and run with your own files

# Generalities

Each analysis is hosted in a single directory, for example `FCCeeAnalyses/ZH_Zmumu/` and contains the same kind of files, please use the same naming convention for all analysis.

1. `analysis.py` : This class that is used to define the list of analysers and filters to run on as well as the output variables.
2. `preSel.py` : This configuration file is used to define how to run the `analysis.py` . It contains the list of samples, the number of CPUs, the fraction of the original sample to process and the base directory for the yaml files (that contains the informations about the samples). This will run the `analysis.py` with a common code `bin/runDataFrame.py` (this last file is common to all analyses and should not be touched).
3. `finalSel.py` : This configuration file contains the final selections and it runs over the locally produced flat ntuples from the `preSel.py` . It contains a link to the `procDict.json` for getting cross section etc...(this might be removed later to include everything in the yaml, closer to the sample), the list of processes, the number of CPU, the cut list, and the variables (that will be both written in a `TTree` and in the form of `TH1` properly normalised to an integrated luminosity of 1pb-1.
4. `plots.py` : This configuration files is used to select the final selections from running `finalSel.py` to plot. Informations about how to merge processes, write some extra text, normalise to a given integrated luminosity etc... For the moment it is possible to only plot one signal at the time, but several backgrounds.

python myAnalysis/preSel.py

will run analysis.py on the datasets defined in preSel.py and available in the common database

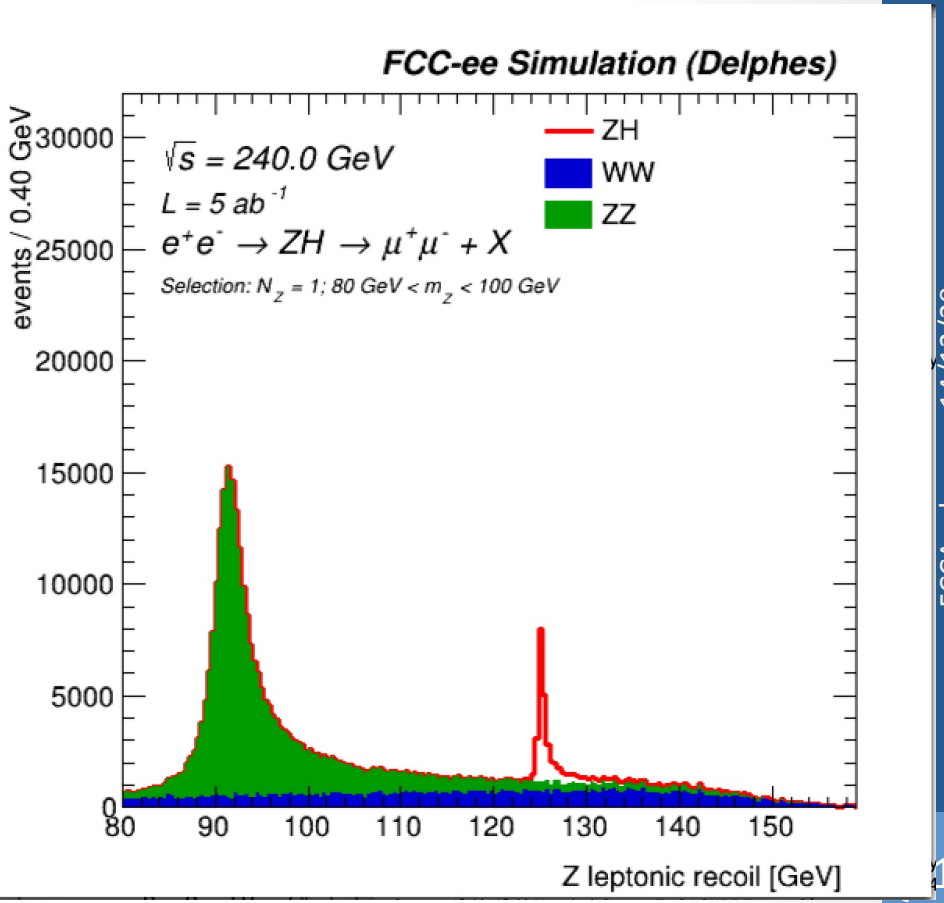
python myAnalysis/analysis.py /myfiles/file.root or /myfiles/*.root

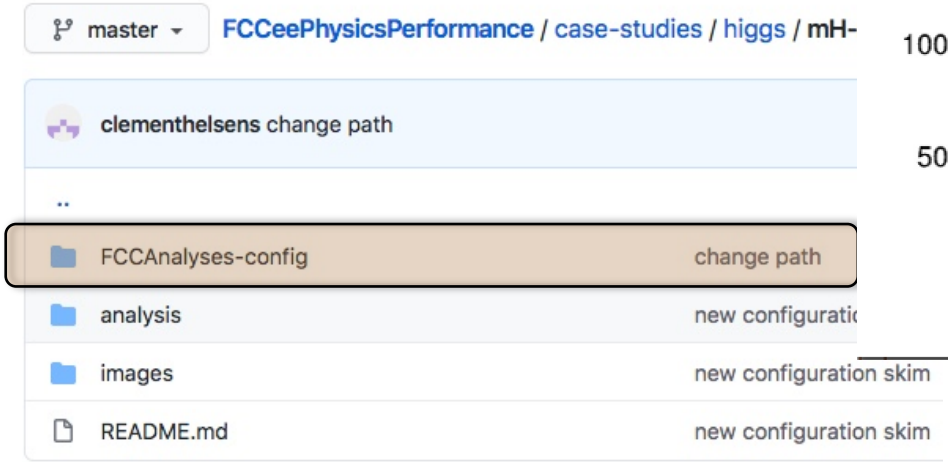
will run analysis.py on all the input files specified (user responsibility to validate that the files are OK

# Examples

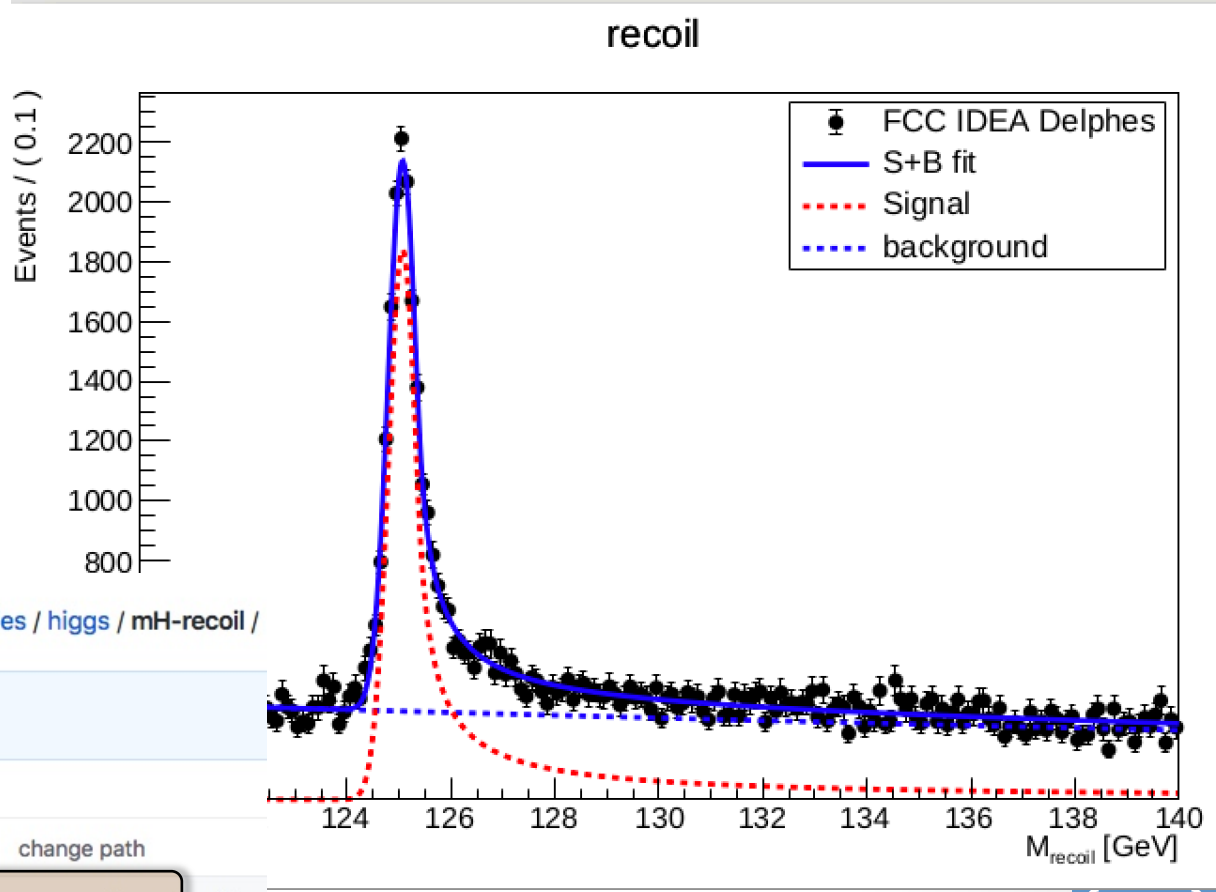
# ZH threshold

- Used the common delphes IDEA track covariance samples I produced
- Processed them through FCCAnalyses machinery with a Z->mumu selection
- And within very little time make this recoil plot
- Anybody with read access to eos can redo those plots in a matter of minutes



- Then fit the recoil with custom roofit code
  - this part has to be written by analysers and goes in the analysis
- Fitted mass:
  - 125.07 +/- 0.00357 GeV




master ▾ FCCeePhysicsPerformance / case-studies / higgs / mH-recoil /

clementhelsens change path

| | |
|---|---|
| .. | |
| FCCAnalyses-config | change path |
| analysis | new configuration skim |
| images | new configuration skim |
| README.md | new configuration skim |

# Flavour physics at the Z pole

# Define variables and filters: MC

analysis.py

```
.Alias("MCRecoAssociations0", "MCRecoAssociations#0.index")
.Alias("MCRecoAssociations1", "MCRecoAssociations#1.index")
.Alias("Particle0", "Particle#0.index")

 .Filter("filterMC_pdgID(541, true)(Particle)==true")
.Filter("filterMC_pdgID(541, true)(Particle)==false")

.Define("MC_px",          "getMC_px(Particle)")
.Define("MC_py",          "getMC_py(Particle)")
.Define("MC_pz",          "getMC_pz(Particle)")
.Define("MC_p",           "getMC_p(Particle)")
.Define("MC_e",           "getMC_e(Particle)")
.Define("MC_pdg",         "getMC_pdg(Particle)")
.Define("MC_charge",      "getMC_charge(Particle)")
.Define("MC_mass",        "getMC_mass(Particle)")
.Define("MC_status",      "getMC_genStatus(Particle)")
.Define("MC_vertex_x",    "getMC_vertex_x(Particle)")
.Define("MC_vertex_y",    "getMC_vertex_y(Particle)")
.Define("MC_vertex_z",    "getMC_vertex_z(Particle)")
```

Needed by RDF to understands #

Filter if event contains a Bc hadron or not, This will create an exclusive sample or inclusive -exclusive

Define the MC quantities to keep

# Define variables and filters: Reco

analysis.py

```
.Define("RP_p",          "getRP_p(ReconstructedParticles)")
.Define("RP_e",          "getRP_e(ReconstructedParticles)")
.Define("RP_px",         "getRP_px(ReconstructedParticles)")
.Define("RP_py",         "getRP_py(ReconstructedParticles)")
.Define("RP_pz",         "getRP_pz(ReconstructedParticles)")
.Define("RP_charge",     "getRP_charge(ReconstructedParticles)")
.Define("RP_mass",       "getRP_mass(ReconstructedParticles)")
```

Access reconstructed particles quantities

```
#.Define("RP_TRK_D0",      "getRP2TRK_D0(ReconstructedParticles, EFlowTrack_1)")
#.Define("RP_TRK_Z0",      "getRP2TRK_Z0(ReconstructedParticles, EFlowTrack_1)")
```

Can get the track associated to them

```
.Define('RP_MC_index',           "getRP2MC_index(MCRecoAssociations0,MCRecoAssociations1,ReconstructedParticles)")
```

Build a collection of size Reco that contains the index in the MC_* columns for the associated MC particle

```
.Define('RP_MC_parentindex',     "getMC_parentid(RP_MC_index,Particle, Particle0)")
```

Use the column defined above and build a new column of MC parents index for each Reco particles

```
.Define('RP_MC_grandparentindex', "getMC_parentid(RP_MC_parentindex,Particle, Particle0)")
```

Use the column defined above and build a new column of MC grand parents index for each Reco particles

# Define variables and filters: Event

analysis.py

```
.Define('EVT_thrust',       'minimize_thrust("Minuit2","Migrad")(RP_px, RP_py, RP_pz)')
.Define('EVT_thrust_val',   'EVT_thrust.at(0)')
.Define('EVT_thrust_x',     'EVT_thrust.at(1)')
.Define('EVT_thrust_x_err','EVT_thrust.at(2)')
.Define('EVT_thrust_y',     'EVT_thrust.at(3)')
.Define('EVT_thrust_y_err','EVT_thrust.at(4)')
.Define('EVT_thrust_z',     'EVT_thrust.at(5)')
.Define('EVT_thrust_z_err','EVT_thrust.at(6)')

.Define('EVT_sphericity',       'minimize_sphericity("Minuit2","Migrad")(RP_px, RP_py, RP_pz)')
.Define('EVT_sphericity_val',   'EVT_sphericity.at(0)')
.Define('EVT_sphericity_x',     'EVT_sphericity.at(1)')
.Define('EVT_sphericity_x_err','EVT_sphericity.at(2)')
.Define('EVT_sphericity_y',     'EVT_sphericity.at(3)')
.Define('EVT_sphericity_y_err','EVT_sphericity.at(4)')
.Define('EVT_sphericity_z',     'EVT_sphericity.at(5)')
.Define('EVT_sphericity_z_err','EVT_sphericity.at(6)')
```

Call the minimisation for the thrust and sphericity and define the relevant columns

EVT_thrust/sphericity is a vector of size 7
Value/x-axis/x-axis-error/y-axis/y-axis-error/z-axis/z-axis-error

# Define variables and filters: Event

analysis.py

```
.Define('RP_thrustangle',      'axisCosTheta(EVT_thrust, RP_px, RP_py, RP_pz)')
.Define('RP_sphericityangle',  'axisCosTheta(EVT_sphericity, RP_px, RP_py, RP_pz)')
```

cosTheta between an axis and the reconstructed particles

```
.Define('EVT_thrusthemis0_q_kappa1',  'getAxisCharge(0, 1)(RP_thrustangle, RP_charge, RP_px, RP_py, RP_pz)')
.Define('EVT_thrusthemis1_q_kappa1',  'getAxisCharge(1, 1)(RP_thrustangle, RP_charge, RP_px, RP_py, RP_pz)')
```

Charge of a given hemisphere
(defined by the sign of cosTheta)

```
.Define('EVT_thrusthemis0_n',         'getAxisN(0)(RP_thrustangle, RP_charge)')
.Define('EVT_thrusthemis1_n',         'getAxisN(1)(RP_thrustangle, RP_charge)')
```

Number of particles (total/charged/neutral)
in each hemisphere

```
.Define('EVT_thrusthemis0_e',         'getAxisEnergy(0)(RP_thrustangle, RP_charge, RP_e)')
.Define('EVT_thrusthemis1_e',         'getAxisEnergy(1)(RP_thrustangle, RP_charge, RP_e)')
```

Energy sum of particles (total/charged/neutral)
in each hemisphere

```
.Define('EVT_thrutshemis0_ncharged',  'EVT_thrusthemis0_n.at(1)')
.Define('EVT_thrutshemis0_nneutral',  'EVT_thrusthemis0_n.at(2)')
```

Number of particles (charged/neutral) in hemisphere with cosTheta<0

```
.Define('EVT_thrutshemis0_echarged',  'EVT_thrusthemis0_e.at(1)')
.Define('EVT_thrutshemis0_eneutral',  'EVT_thrusthemis0_e.at(2)')
```

Energy sum of particles (charged/neutral) in hemisphere with cosTheta<0

```
.Define('EVT_thrutshemis1_ncharged',  'EVT_thrusthemis1_n.at(1)')
.Define('EVT_thrutshemis1_nneutral',  'EVT_thrusthemis1_n.at(2)')
```

Number of particles (charged/neutral) in hemisphere with cosTheta>0

```
.Define('EVT_thrutshemis1_echarged',  'EVT_thrusthemis1_e.at(1)')
.Define('EVT_thrutshemis1_eneutral',  'EVT_thrusthemis1_e.at(2)')
```

Energy sum of particles (charged/neutral) in hemisphere with cosTheta>0

```
.Define('EVT_thrutshemis_emax',  'if (EVT_thrusthemis0_e.at(0)>EVT_thrusthemis1_e.at(0)) return EVT_thrusthemis0_e.at(0); else return EVT_thrusthemis1_e.at(0);')
.Define('EVT_thrutshemis_emin',  'if (EVT_thrusthemis0_e.at(0)>EVT_thrusthemis1_e.at(0)) return EVT_thrusthemis1_e.at(0); else return EVT_thrusthemis0_e.at(0);')
```

Define the hemisphere with minimum/maximum energies

# Define variables and filters: Run

analysis.py

```
# select branches for output file
branchList = ROOT.vector('string')()
for branchName in [

        "MC_px","MC_py","MC_pz","MC_p","MC_e","MC_pdg","MC_charge","MC_mass","MC_status","MC_vertex_x","MC_vertex_y","MC_vertex_z",

        "EVT_thrust_x","EVT_thrust_y","EVT_thrust_z","EVT_thrust_val","EVT_thrusthemis0_q_kappa1","EVT_thrusthemis1_q_kappa1",
        "EVT_thrutshemis0_ncharged","EVT_thrutshemis1_ncharged","EVT_thrutshemis0_nneutral","EVT_thrutshemis1_nneutral",
        "EVT_thrutshemis0_echarged","EVT_thrutshemis1_echarged","EVT_thrutshemis0_eneutral","EVT_thrutshemis1_eneutral",
        "EVT_thrutshemis_emax","EVT_thrutshemis_emin",

        "EVT_sphericity_x","EVT_sphericity_y","EVT_sphericity_z","EVT_sphericity_val",

        "RP_thrustangle","RP_sphericityangle","RP_p","RP_px","RP_py","RP_pz","RP_charge","RP_mass",

        "RP_MC_index","RP_MC_parentindex","RP_MC_grandparentindex",

        ]:
    branchList.push_back(branchName)
```

Defined the columns to keep

```
#opts = ROOT.RDF.RSnapshotOptions()
#opts.fCompressionAlgorithm = ROOT.ROOT.kLZ4
#opts.fCompressionLevel = 3
#opts.fAutoFlush = -1024*1024*branchList.size()
#df2.Snapshot("events", self.outname, branchList, opts)
df2.Snapshot("events", self.outname, branchList)
```

And run

# preSel configuration

preSel.py

```
#python examples/FCCee/flavour/generic-analysis/preSel.py

from config.common_defaults import deffccdicts
import os

basedir=os.path.join(os.getenv('FCCDICTSDIR', deffccdicts), '') + "yaml/FCCee/fcc_tmp_v02/"
outdir="outputs/FCCee/flavour/generic-analysis/"

import multiprocessing
NUM_CPUS = int(multiprocessing.cpu_count()-2)
process_list=[
    'p8_ee_Zbb_ecm91_EvtGen_Bc2TauNuTAUHADNU',
    'p8_ee_Zcc_ecm91',
    'p8_ee_Zuds_ecm91'
]

#output_list=['p8_ee_Zbb_ecm91']
output_list=[]

fraction=1.0

import config.runDataFrame as rdf
myana=rdf.runDataFrame(basedir,process_list, outlist=output_list)
myana.run(ncpu=NUM_CPUS,fraction=fraction,outDir=outdir)
```

To run with the common production, files are validated etc…

Define here the process list, names are in the Database

Possibility to change the ouptut name, can be used when filtering a more inclusive sample

```
.Filter("filterMC_pdgID(541, true)(Particle)==true")
.Filter("filterMC_pdgID(541, true)(Particle)==false")
```

# finalSel configuration

finaSel.py

```
###Input directory where the files produced at the pre-selection level are
#baseDir  = "outputs/FCCee/flavour/generic-analysis/"
baseDir  = "/eos/experiment/fcc/ee/tmp/flatntuples/Z_Zbb_Bc2TauNu/"

###Link to the dictonary that contains all the cross section informations etc...
#procDict = os.path.join(os.getenv('FCCDICTSDIR', deffccdicts), '') + "FCCee_procDict_fcc_tmp_v02.json"
#procDict ='https://fcc-physics-events.web.cern.ch/fcc-physics-events/sharedFiles/FCCee_procDict_fcc_v02.json'
procDict = {
    "p8_ee_Zbb_ecm91": {"numberOfEvents": 19830000, "sumOfWeights": 19830000, "crossSection":6645.46, "kfactor": 1.0, "matchingEfficiency": 1.0},
    "p8_ee_Zbb_ecm91_EvtGen_Bc2TauNuTAUHADNU": {"numberOfEvents": 14789, "sumOfWeights": 14789, "crossSection": 6645.46*7.9e-5*0.0236*0.098, "kfactor": 1.0, "matchingEfficiency": 1.0},
    "p8_ee_Zcc_ecm91": {"numberOfEvents": 1000000, "sumOfWeights": 1000000, "crossSection": 5215.46, "kfactor": 1.0, "matchingEfficiency": 1.0},
    "p8_ee_Zuds_ecm91": {"numberOfEvents": 1000000, "sumOfWeights": 1000000, "crossSection": 18616.5, "kfactor": 1.0, "matchingEfficiency": 1.0}
}

process_list=['p8_ee_Zbb_ecm91_EvtGen_Bc2TauNuTAUHADNU','p8_ee_Zbb_ecm91','p8_ee_Zcc_ecm91','p8_ee_Zuds_ecm91']

define_list={
    "EVT_Ediff":"EVT_thrutshemis_emax-EVT_thrutshemis_emin",

    "EVT_Echarged_max":"if (EVT_thrutshemis0_echarged>EVT_thrutshemis1_echarged) return EVT_thrutshemis0_echarged; else return EVT_thrutshemis1_echarged;",
    "EVT_Echarged_min":"if (EVT_thrutshemis0_echarged>EVT_thrutshemis1_echarged) return EVT_thrutshemis1_echarged; else return EVT_thrutshemis0_echarged;",
    "EVT_Eneutral_max":"if (EVT_thrutshemis0_eneutral>EVT_thrutshemis1_eneutral) return EVT_thrutshemis0_eneutral; else return EVT_thrutshemis1_eneutral;",
    "EVT_Eneutral_min":"if (EVT_thrutshemis0_eneutral>EVT_thrutshemis1_eneutral) return EVT_thrutshemis1_eneutral; else return EVT_thrutshemis0_eneutral;",

    "EVT_Ncharged_max":"if (EVT_thrutshemis0_ncharged>EVT_thrutshemis1_ncharged) return EVT_thrutshemis0_ncharged; else return EVT_thrutshemis1_ncharged;",
    "EVT_Ncharged_min":"if (EVT_thrutshemis0_ncharged>EVT_thrutshemis1_ncharged) return EVT_thrutshemis1_ncharged; else return EVT_thrutshemis0_ncharged;",
    "EVT_Nneutral_max":"if (EVT_thrutshemis0_nneutral>EVT_thrutshemis1_nneutral) return EVT_thrutshemis0_nneutral; else return EVT_thrutshemis1_nneutral;",
    "EVT_Nneutral_min":"if (EVT_thrutshemis0_nneutral>EVT_thrutshemis1_nneutral) return EVT_thrutshemis1_nneutral; else return EVT_thrutshemis0_nneutral;",

}
```

Using common procDict or custom
Possibility to define new variables

```
ROOT.gInterpreter.Declare("""
bool myFilter(ROOT::VecOps::RVec<float> mass) {
    if (mass.size()<2) return false;
    for (size_t i = 0; i < mass.size(); ++i) {
       if (mass.at(i)<80. || mass.at(i)>100.)
        return false;
}
    return true;
}
""")
```

# finalSel configuration

finaSel.py

```
###Dictionnay of the list of cuts. The key is the name of the selection that will be added to the output file
cut_list = {"sel0":"RP_p.size()>0",
            "sel1":"EVT_thrutshemis_emax<48. && EVT_thrutshemis_emin<35. && EVT_Ediff>10.",
            "sel2":"EVT_thrutshemis_emax<45. && EVT_thrutshemis_emin<25. && EVT_Ediff>15. && EVT_Echarged_min<20. && EVT_Nneutral_mi
            "sel3":"EVT_thrutshemis_emin<10.",
            }

###Dictionary for the ouput variable/hitograms. The key is the name of the variable in the output files. "name" is the name of the v
he histogram, "bin" the number of bins of the histogram, "xmin" the minimum x-axis value and "xmax" the maximum x-axis value.
variables = {

    "EVT_thrust_val":{"name":"EVT_thrust_val","title":"Event Thrust","bin":100,"xmin":0.4,"xmax":1.},
    "EVT_thrusthemis_emax":{"name":"EVT_thrutshemis_emax","title":"Hemisphere energy (max) [GeV]","bin":120,"xmin":0.,"xmax":60},
    "EVT_thrusthemis_emin":{"name":"EVT_thrutshemis_emin","title":"Hemisphere energy (min) [GeV]","bin":120,"xmin":0.,"xmax":60},
    "EVT_Ediff":{"name":"EVT_Ediff","title":"Hemisphere energy difference [GeV]","bin":120,"xmin":0.,"xmax":60},
    "EVT_Echarged_max":{"name":"EVT_Echarged_max","title":"Hemisphere charged energy (max) [GeV]","bin":120,"xmin":0.,"xmax":60},
    "EVT_Echarged_min":{"name":"EVT_Echarged_min","title":"Hemisphere charged energy (min) [GeV]","bin":120,"xmin":0.,"xmax":60},
    "EVT_Eneutral_max":{"name":"EVT_Eneutral_max","title":"Hemisphere neutral energy (max) [GeV]","bin":120,"xmin":0.,"xmax":60},
    "EVT_Eneutral_min":{"name":"EVT_Eneutral_min","title":"Hemisphere neutral energy (min) [GeV]","bin":120,"xmin":0.,"xmax":60},
    "EVT_Ncharged_max":{"name":"EVT_Ncharged_max","title":"Hemisphere charged multiplicity (max)","bin":25,"xmin":0.,"xmax":25},
    "EVT_Ncharged_min":{"name":"EVT_Ncharged_min","title":"Hemisphere charged multiplicity (min)","bin":25,"xmin":0.,"xmax":25},
    "EVT_Nneutral_max":{"name":"EVT_Nneutral_max","title":"Hemisphere neutral multiplicity (max)","bin":25,"xmin":0.,"xmax":25},
    "EVT_Nneutral_min":{"name":"EVT_Nneutral_min","title":"Hemisphere neutral multiplicity (min)","bin":25,"xmin":0.,"xmax":25}
}

###Number of CPUs to use
NUM_CPUS = 4

###This part is standard to all analyses
import config.runDataFrameFinal as rdf
#myana=rdf.runDataFrameFinal(baseDir,procDict,process_list,cut_list,variables)
myana=rdf.runDataFrameFinal(baseDir,procDict,process_list,cut_list,variables,defines=define_list)
myana.run(ncpu=NUM_CPUS, doTree=False)
```

```
ROOT.gInterpreter.Declare("""
bool myFilter(ROOT::VecOps::RVec<float> mass) {
    if (mass.size()<2) return false;
    for (size_t i = 0; i < mass.size(); ++i) {
       if (mass.at(i)<80. || mass.at(i)>100.)
        return false;
}
    return true;
}
""")
```

```
"sel3":"myFilter(zed_leptonic_m)"
```

# plots configuration

plots.py

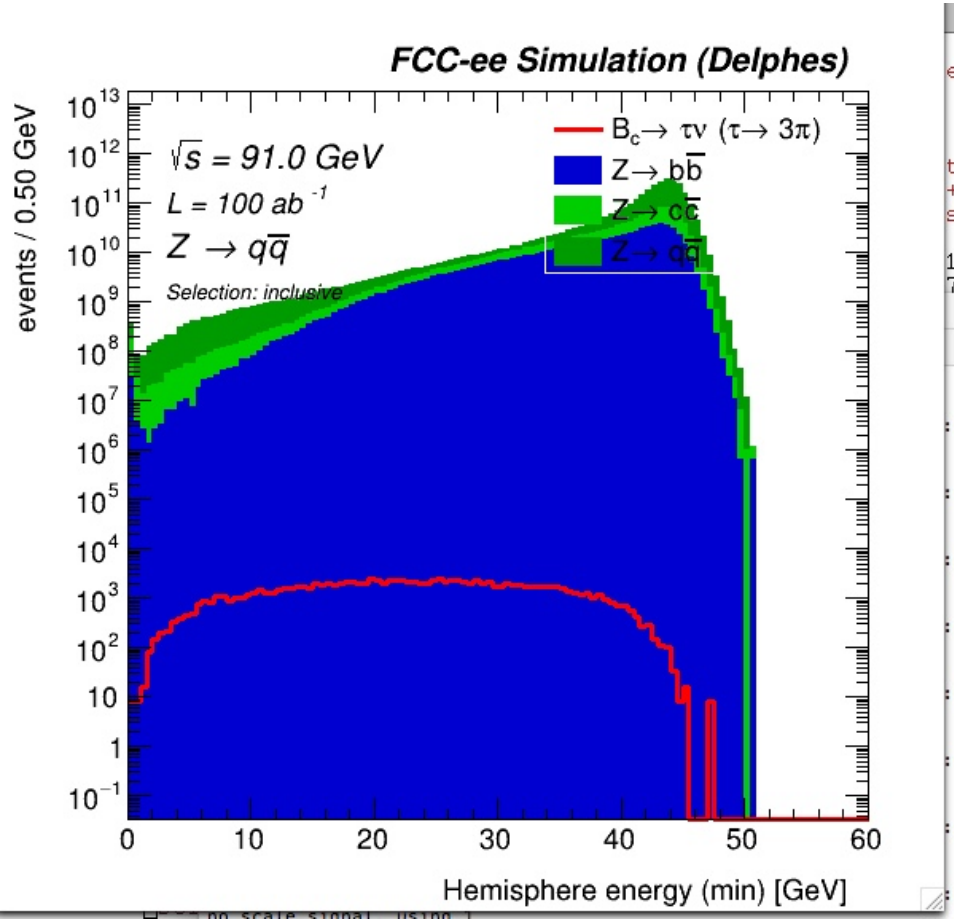
```
# global parameters
intLumi        = 100000000. #in pb-1
ana_tex        = "Z #rightarrow q#bar{q}"
delphesVersion = "3.4.2"
energy         = 91.0
collider       = "FCC-ee"
inputDir       = "/eos/experiment/fcc/ee/tmp/flatntuples/Z_Zbb_Bc2TauNu/"
formats        = ['png','pdf']
yaxis          = ['lin','log']
stacksig       = ['stack','nostack']
outdir         = 'outputs/FCCee/flavour/generic-analysis/plots/'

variables = [ "EVT_thrust_val",
              "EVT_thrusthemis_emax",
              "EVT_thrusthemis_emin",
              "EVT_Ediff",
              "EVT_Echarged_max",
              "EVT_Echarged_min",
              "EVT_Eneutral_max",
              "EVT_Eneutral_min",

              "EVT_Ncharged_max",
              "EVT_Ncharged_min",
              "EVT_Nneutral_max",
              "EVT_Nneutral_min"

              ]
```

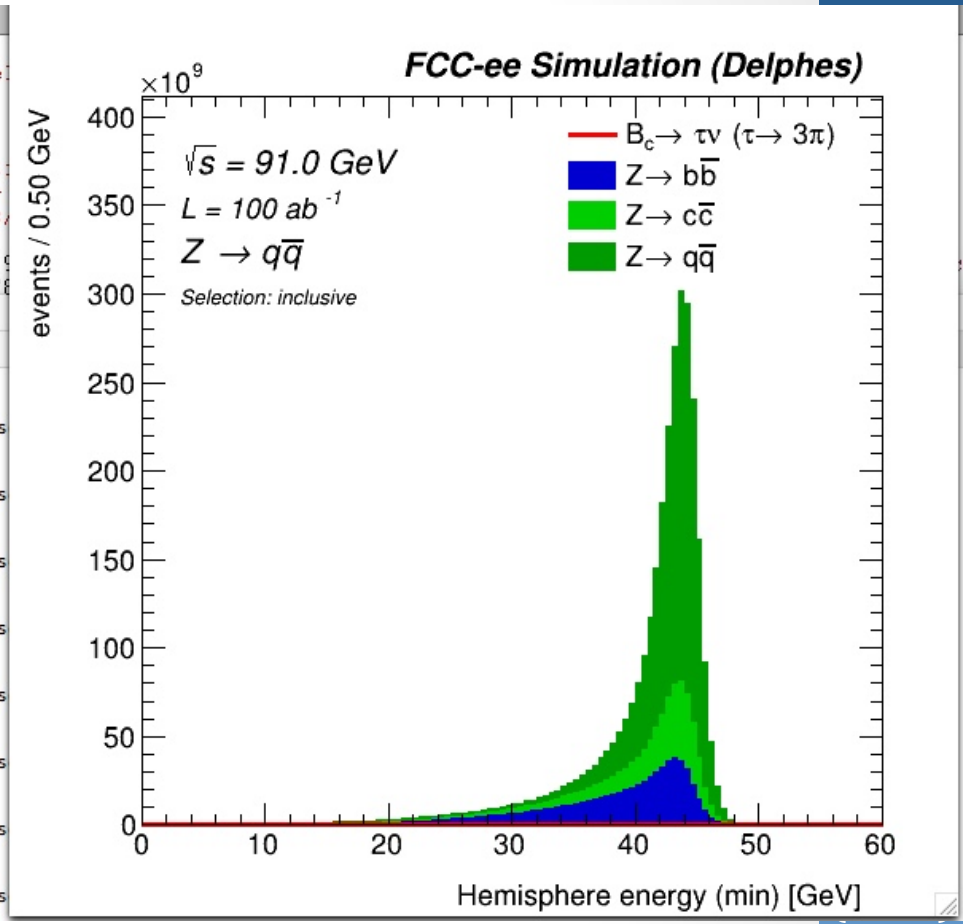
```
###Dictonnary with the analysis name as a key, and the list of selections to b
selections = {}
selections['Flavour']   = ["sel0","sel1","sel2","sel3"]

extralabel = {}
extralabel['sel0'] = "Selection: inclusive"
extralabel['sel1'] = "Selection: with cuts"
extralabel['sel2'] = "Selection: with tighter cuts"
extralabel['sel3'] = "Selection: with even tighter cuts"
```

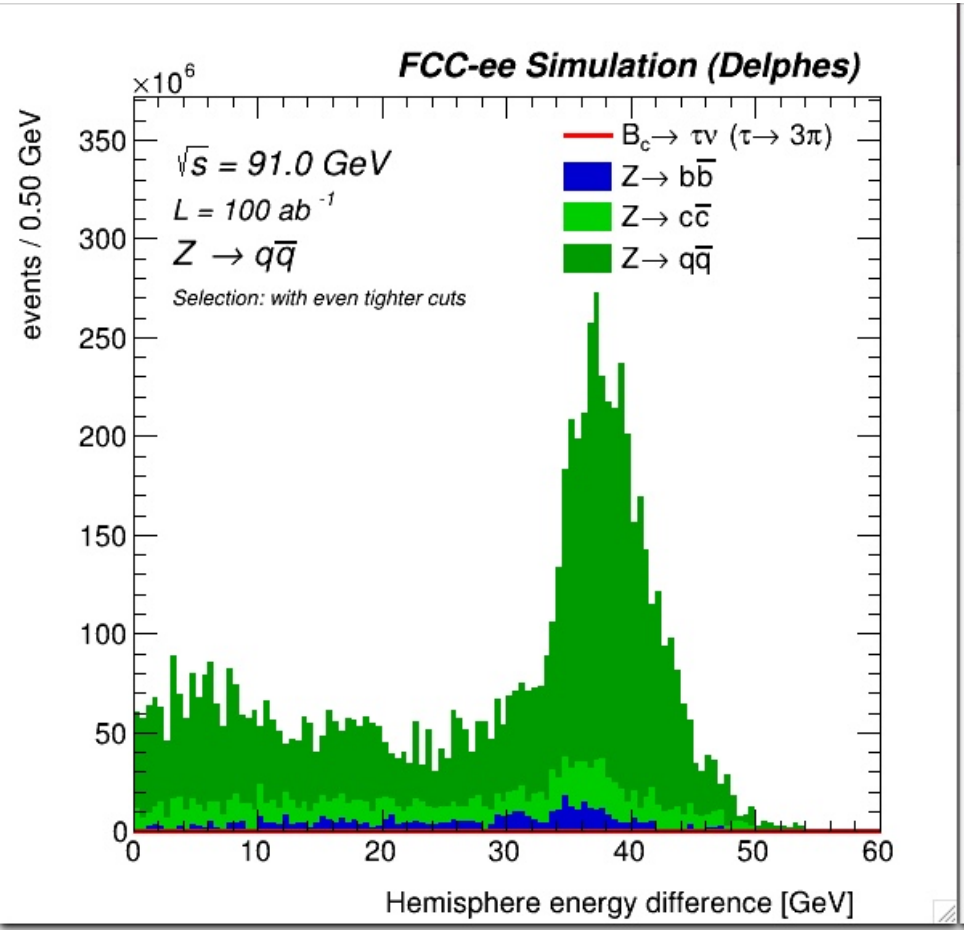
```
colors = {}
colors['Z_flavour'] = ROOT.kRed
colors['Z_bb'] = ROOT.kBlue+1
colors['Z_cc'] = ROOT.kGreen+1
colors['Z_uds'] = ROOT.kGreen+2

plots = {}
plots['Flavour'] = {'signal':{'Z_flavour':['p8_ee_Zbb_ecm91_EvtGen_Bc2TauNuTAUHADNU']},
                    'backgrounds':{'Z_bb':['p8_ee_Zbb_ecm91'],
                                   'Z_cc':['p8_ee_Zcc_ecm91'],
                                   'Z_uds':['p8_ee_Zuds_ecm91']}
          }

legend = {}
legend['Z_flavour'] = 'B_{c}#rightarrow #tau#nu (#tau#rightarrow 3#pi)'
legend['Z_bb'] = 'Z#rightarrow b#bar{b}'
legend['Z_cc'] = 'Z#rightarrow c#bar{c}'
legend['Z_uds'] = 'Z#rightarrow q#bar{q}'
```

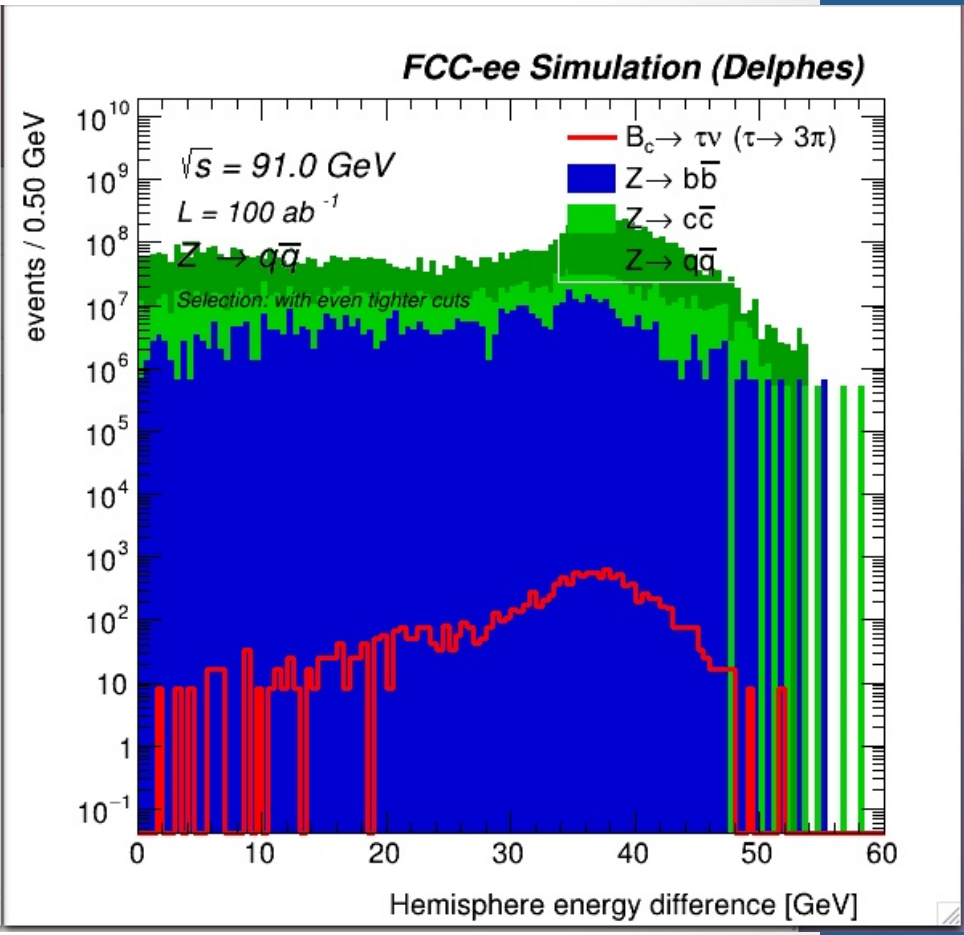
Common plot style will be supported here. But users could write their own as well

Plan to write a common plotting code independent of ROOT


FCC-ee Simulation (Delphes)
$\sqrt{s}$ = 91.0 GeV
L = 100 ab$^{-1}$
$Z \rightarrow q\bar{q}$
Selection: inclusive
$B_c \rightarrow \tau\nu$ ($\tau \rightarrow 3\pi$)
$Z \rightarrow b\bar{b}$
$Z \rightarrow c\bar{c}$
$Z \rightarrow q\bar{q}$
events / 0.50 GeV
Hemisphere energy (min) [GeV]


FCC-ee Simulation (Delphes)
$\times 10^9$
$\sqrt{s}$ = 91.0 GeV
L = 100 ab$^{-1}$
$Z \rightarrow q\bar{q}$
Selection: inclusive
$B_c \rightarrow \tau\nu$ ($\tau \rightarrow 3\pi$)
$Z \rightarrow b\bar{b}$
$Z \rightarrow c\bar{c}$
$Z \rightarrow q\bar{q}$
events / 0.50 GeV
Hemisphere energy (min) [GeV]


FCC-ee Simulation (Delphes)
×10^6
√s = 91.0 GeV
L = 100 ab^-1
Z → qq̄
Selection: with even tighter cuts
B_c→ τν (τ→ 3π)
Z→ bb̄
Z→ cc̄
Z→ qq̄
events / 0.50 GeV
Hemisphere energy difference [GeV]


FCC-ee Simulation (Delphes)
√s = 91.0 GeV
L = 100 ab^-1
Z → qq̄
Selection: with even tighter cuts
B_c→ τν (τ→ 3π)
Z→ bb̄
Z→ cc̄
Z→ qq̄
events / 0.50 GeV
Hemisphere energy difference [GeV]

# FCCAnalysis: available algorithms

- Available functionality (see doxygen)
  - Thrust axis and thrust value
  - Sphericity axis and value
  - cos(theta) between an axis and particles
  - Weighted charge of all particles with cos(theta) axis > or <0
  - Preliminary jet clustering
  - Truth history from any MC particle of a given status
  - Tagging 2 body MC decays (for example H->gg, Z->bb)
  - Etc…
- Planned functionalities for the common analysers
  - Acceptance efficiencies
  - Parametric PID
  - Pi0 identification
  - Etc…
- Planning to have a dedicated uproot/awkward structure to easily run flavour exclusive decays, etc, etc...

# Summary/Plans

- Easy to use analysis code to produce flat ntuples based on Root DataFrame ready for you to run analyses, please join and have fun
- Did not talked about FCC-hh but all this code can be used there as well (or in ATLAS/CMS?)

- Plans
  - Adding uproot and awkward from pure python or C++
  - Etc…