FNAL Column Analysis Tools

Lindsey / Nick

12 December 2018

What is this package?

- It's a prototyping area for tools to use in the columnar analysis pattern
  - Typically biased towards the "striped" database implemented by Igor Mandrichenko
    - i.e. you get a 20000xN_cols chunk of data as numpy arrays and are tasked to process it as accurately and quickly as possible
  - Heavy use of awkward array from Jim Pivarski
    - Allows expressive/compact representation of data and operations upon it
  - Try to make things as close to "standard" analysis as possible, just no loops
    - Use lightweight wrappers where needed to make operations on data expressive to a physicist
  - Large suite of tools for doing corrections typical to analysis
    - Scale factor and smearing lookups from root, json, b-tag csv, JEC, JER (last two are work-in-progress)

Example: Z-peak with Scale Factors

- https://github.com/lgray/CoffeaMaker/blob/master/zpeak_lg.ipynb
  - Selected electron and muon Z candidates, trigger match (optional), plot results with scale factors
  - 100kHz per stripe, ~10 MHz with all workers
    - Sufficiently fast for a reasonably complex selection

```
electronCols = PhysicalColumnGroup(events,"Electron","p4","charge","tightID")
electrons_new = jaggedFromColumnGroup(electronCols)

muonCols = PhysicalColumnGroup(events,"Muon","p4","charge","tightID")
muons_new = jaggedFromColumnGroup(muonCols)

electrons_new['SF'] = weights_eval["eleScaleFactor_TightId_POG"](electrons_new.eta,
                                                                 electrons_new.pt)
muons_new['SF'] = weights_eval["muScaleFactor_TightId_Iso"](np.absolute(muons_new.eta),
                                                            muons_new.pt)

electrons_trig = electrons_new#[events_triggered]
muons_trig = muons_new#[events_triggered]

electron_selection = ( (electrons_trig.pt > 20) &
                       (np.abs(electrons_trig.eta) < 2.5) &
                       (electrons_trig.tightID > 0) )
muon_selection = ( (muons_trig.pt > 20) &
                   (np.abs(muons_trig.eta) < 2.4) &
                   (muons_trig.tightID > 0) )
```

How do you get this package?

- pip install fnal-column-analysis-tools
- import fnal_column_analysis_tools
- Three sub-packages:
  - lookup_tools : read in and apply scale factors / etc.
  - analysis_objects : wrapper for turning groups of numpy columns into analysis objects
  - striped : small conversion / management layer for getting data from striped

Current Status of Lookup Tools (dev6)

- In fnal-column-analysis-tools you can read in:
  - Histograms in json and root file formats
    - Evaluation of histogram lookup runs at 50 MHz
    - Over/underflows are clamped to the highest/lowest bin
  - b-tagging scale factors from BTV csv files
    - Evaluation runs at ~2 MHz due to hitting a python loop
    - Working on fixing this, performance should improve drastically

histogram lookup

```
In [11]: #apply the previously loaded scale factors for the "muons" and "electrons"
         #and put them in the objects
         # we need the |eta| of the muons for their scale factors, maybe we should add this to TLorentzVector?
         %timeit muons["SF"] = weights_eval["muScaleFactor_MediumId_Iso"](np.abs(muons.eta),muons.pt)
         %timeit electrons["SF"] = weights_eval["eleScaleFactor_MediumId_MVA_POG"](electrons.eta,electrons.pt)

         10 loops, best of 3: 22.3 ms per loop
         100 loops, best of 3: 15.8 ms per loop
```

<—1e6 events

btagSF lookup

```
In [7]: print(eta.size)
        a = {}
        a["out"] = weights_eval['CSVv2_0_comb_central_0'](np.abs(eta), pt, x)
        %timeit a["out"] = weights_eval['CSVv2_0_comb_central_0'](np.abs(eta), pt, x)

        1000000
        1 loop, best of 3: 444 ms per loop
```

What's in Development?

- JEC calculation/application
  - Waiting on new version of awkward-array
    - Need to have numpy-like indexing trick to make extracting list of corrected jet energies straightforward
    - Implementation mostly using numpy, should be fast
- Integrate awkward v0.7 features into candidate
  - Nested pairs for generator and cross-cleaning matching
    - Wrap into more user-friendly functions